Modeling user-system data exchange to design adaptive interfaces

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

ABSTRACT

The wide spread of mobile devices in the consumer market has posed a number of new issues in the design of internet applications and their user interfaces. In particular, applications need to adapt their interaction modalities to different portable devices, like cellular phones and handhelds.

In this paper we address the problem of defining models and techniques for designing internet based applications that automatically adapt to different mobile devices. First, we define a formal model that allows for specifying the interaction in a way that is abstract enough to be decoupled from the presentation layer, which is to be adapted to different contexts. The model is mainly based on the idea of describing the user interaction in terms of elementary actions, (Atomic Interaction Units, AIU) that capture the user interaction through the data s/he exchanges with the system. Then, we provide a formal device characterization showing how to effectively implements the AIUs in a multidevice context.

Keywords: Adaptive interfaces, mobile devices, internet based applications

1. INTRODUCTION

In this paper we deal with the problem of designing user interfaces for internet based applications that can run on multiple heterogeneous devices while ensuring usability. The advent of mobile devices and the consequent diffusion of computing capabilities out of standard contexts, open new and challenging questions. The era of standard situated PCs is over; nowadays users own multiple different computing appliances that can be easily transported and that, consequently, operate in multiple changing environments. Moreover, the large diffusion of such kind of devices among society means that they are penetrating parts of population never reached before. In order to address this issue the next generation of internet based application should be designed for adapting to a very large spectrum of different characteristics. This is why the research is faced with the problem of finding models and techniques to render applications aware of and adaptable to:

1. Devices - many different appliances (e.g., cellphones, PDAs, PCs, kiosks)
2. Environments - many different conditions/situations (e.g. noisy room, low light, moving person)
3. Users - many different users in terms of their preferences, their capabilities (e.g., users with special needs), their goals

In this paper we focus on the first issue, i.e., adapting the application to different, mobile devices; we provide some hints on how to take into account the other two parameters, environment and user, but deepening these aspects is out of the scope of this paper. To deal with the above problem we need to deal with many different issues:

- architectures that support heterogeneous devices;
- models to exchange and share data;
- techniques for building effective user interfaces.

Our focus is on user interfaces and our current approach is mainly based on the idea of modelling 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 interface 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.

1.1 The application scenario

In this section we would like to clarify the context in which our proposal takes place. We want to deal with simple internet based applications/services, like reserving an hotel room, finding a restaurant, or booking a flight seat. It is
out of the scope of our approach to redesign generic web sites making them accessible through different devices or implementing complex applications over large information systems. We are focusing on the plethora of simple but useful applications that can run on a cellular phone, on a connected PDA, and so on, providing the user with concise pieces of information and/or simple services.

Concerning the way we choose to attack this problem, we have to consider that a great effort has been spent to find techniques and tools for designing applications that can run on multiple contexts. We describe, in the following, the various adopted approaches, positioning our proposal within them.

A first distinction arises between solutions based on the idea of supporting the design and solutions based on the idea of re-authoring existing applications. It is in fact clear that, along with the need of providing tools that support the designer while building a new application that should run and adapt to heterogeneous contexts, a strong need for tools that convert existing applications in order to let them support new needs exists as well.

The amount of provided automation is another dimension along which current proposals can be classified. The distinction is between applications that are designed once for several prefixed contexts and those that can automatically adapt on-the-fly to the various situations. Even if a sharp separation cannot be done it is quite evident that some proposals tend to be more self-adaptive/dynamic while others embrace a totally static design approach.

Our solution, as it will be explained in the following sections, is mainly oriented to support the designer in his/her work of multi-context application design. The idea is to describe the application in a very abstract way having a server that on the basis of both application and device descriptions automatically produces the user interface.

1.2 Organization of the paper
The rest of the document is structured as follows: in Section 2 we describe some related proposals, in Section 3 we introduce a model that supports the design of adaptive applications: the Atomic Interaction Units model and a working example is provided. Section 4, provides a description of the main device characteristics we want to take into account. Section 5 describes some preliminary results on the AIU implementation, and finally in Section 6 some conclusions and open issues are discussed.

2. 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. It is becoming a more and more relevant issue in user interface research community and several proposals from many research areas have been produced during last times. They all share the common idea of using formal models and well structured processes, in which the generation of the final interface and the application code is just the last step of a composite activity.

Many ideas come from past research in model-based user interface design [7] where the designer is supposed to design an interactive system by editing and manipulating abstract models (e.g. task model) that describe the system’s behavior and where the system is supposed to automatically generate the final application code.

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 (AIOs) that are platform-neutral widgets he can assemble to design the interface in an abstract manner. Different presentation structures can then be generated to allocate the interaction units across many windows. The choice about what concrete interaction units correspond to abstract ones and the generation of different presentation structures permit to optimize the screen real estate and let the user interface nicely match with the interaction capabilities of the current device.

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 modelling. The authors propose a very structured process in which the designer must follow some predefined steps. First he creates a task model. Then he creates a revised task model for each platform, in order to take into account different task structures for the different devices. The system then automatically generate a series of presentation units that the designer fills with abstract interaction units using composition operators. Finally, the real interface is generated, for each considered platform, transforming abstract units into concrete units and translating abstract compositional operators into specific platform interactors. The interesting aspect of this proposal is that the presentation structure is automatically generated from a task model, taking into account temporal constraints specified into it. Screen space optimization is obtained mapping the abstract objects into suitable concrete objects. It is not clear if the designer can also easily operate on the dialog structure in order to generate presentations that nicely match the device’s interactive capabilities.

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 modelling. 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. Finally, he creates a presentation part that permits to give a visual form to the various pages. 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 abstracting on interaction elements and provide 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 modelling approach. 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 (an UML Activity Diagram) in which the nodes are populated with atomic interaction units and the edges represent the transition triggered interacting with atomic interaction units.

3. FORMALIZING THE INTERACTION: THE ATOMIC INTERACTION UNITS

As described in the introduction, the foundation of our proposal is an abstract model able to describe the user interaction with the system describing the basic activities whose composition will produce simple but effective internet based application. As a consequence, we model the information that is exchanged between the user and the system together with the purpose for which such an information is exchanged. Moreover, the interaction modeling foresees different way of presenting the same information, in order to adapt it to different physical means and or channels, like mouse, touch screen, audio, etc. These issues will be tickled in Section 5. 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 interact 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. Moreover, 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, described in the following section.

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.

3.1 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. Moreover, a browsing activity may return some values to the system, e.g., a point returned while browsing an image. The inputting activity is based on two different strategies: filling with free text some fields or choosing among several predefined choices.

According to these strategies we foresee a basic set of atomic interaction units (AIUs), namely, BrowserImage, InteractImage, BrowseText, BrowseMessage, BrowseTable, InteractTable, FillList, SelectChoice, SelectMultipleChoice. Each AIU is characterized by a signature that defines the input it expects and the output it returns. In the following of this section each AIU is described in detail. Note that all the AIUs share a Quit command that allows for leaving the AIU with no effects and returning the null value.

- **BrowserImage**
  
  ```
  (ImageId, ImageDescription, ListOfBrowsingCommands)
  :
  (NULL, elemOfListOfBrowsingCommands)
  ```

  The **BrowserImage** AIU allows for browsing an image; usual facilities of zooming and panning are provided, if possible, by the device. The image description is a two values record: [ImageName, ImageSummary], where ImageName is used as a title during the image presentation and ImageSummary is an image description that can be used when the video channel is not available or disturbed. The ImageSummary can be used considering the environment or the user as well. As an example, assume the user is driving a car or is visually impaired; in such a case, if the audio channel is available the interface can exploit it to deliver the ImageSummary. The ListOfBrowsingCommands is a set of commands oriented towards server side image manipulation (e.g., changing image detail and/or resolution); such commands do not allow to reach any other state than the one hosting the AIU (i.e., they correspond to self-transitions). 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 a leaf of his/her interaction with the system, i.e., to digest some pieces of information presented in a graphical way.

- **InteractImage**
  
  ```
  (ImageId, ImageDescription, ListOfBrowsingCommands)
  :
  (point, NULL, elemOfListOfBrowsingCommands)
  ```

  The **InteractImage** AIU allows the user to interact with the image: click on a point of the image to take some action with the system; the system returns the coordinates of the point and executes the action with the ImageSummary. The ListOfBrowsingCommands is a set of commands oriented towards server side image manipulation (e.g., changing image detail and/or resolution); such commands do not allow to reach any other state than the one hosting the AIU (i.e., they correspond to self-transitions). The null value signals that the user terminated the AIU through the Quit command.
The **BrowseText AIU** is quite similar to the **BrowseImage AIU**; the only difference is that the user can leave the AIU both choosing the Quit button (the AIU returns the null value) or selecting a point (the AIU returns the x,y coordinates of a point) on the image itself.

**USAGE:** the purpose of this AIU is to allow the user to provide the system with an input that is produced through a graphical interaction (e.g., selecting a location on a map).

- **BrowseText**
  
  (TextId, TextDescription, 
  ListOfBrowsingCommands, OKbutton) 
  :
  (NULL, elemOfListOfBrowsingCommands)

The **BrowseText AIU** allows for browsing a large body of text; usual facilities of panning are provided, if possible, by the device. The text description is a two value record: [TextName, TextSummary], where TextName is used as a title during the text presentation and TextSummary 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 such an amount of text is very poor. The ListOfBrowsingCommands is a set of commands oriented towards server side text manipulation (e.g., moving quickly to a text part or expanding some concept); 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 a leaf of his/her interaction with the system, i.e., to digest some pieces of information presented in a textual way.

- **BrowseMessage**
  
  (MessageId, 
  MessageDescription, OKbutton) 
  :
  (NULL, OK)

The **BrowseMessage AIU** allows for browsing a message; usual facilities of panning are provided, if possible, by the device. The MessageDescription is a two value record: [MessageName, MessageSummary], where MessageName is used as a title during the text presentation and MessageSummary 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 such an amount of text is very poor. The ListOfBrowsingCommands is a set of commands oriented towards server side text manipulation (e.g., moving quickly to a text part or expanding some concept); 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 see a message that may contain a question (e.g., do you agree? retry?).

- **BrowseTable**
  
  (TableId, TableDescription, 
  ListOfBrowsingCommands, Mode) 
  :
  (NULL, elemOfListOfBrowsingCommands)

The **BrowseTable AIU** allows for browsing a relational table (with or without BLOBs and CLOBs); usual facilities of panning 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. The parameter Mode has three values:

Full : the table is presented to the user without any omission; Manual : the system allows for altering (client side) the table structure; Automatic: based on the user profile and other parameters the table is reduced (server side, through selection and projection operations).

**USAGE:** the purpose of this AIU is to allow the user to explore a leaf of his/her interaction with the system, i.e., to digest 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:** the purpose of this AIU is to allow the user to provide the system with some input, input that is produced through the interaction with a table, i.e., the user selects a tuple.

- **FillList**
  
  (ListId, ListDescription, 
  ListOfField) 
  :
  (NULL, ListOfField)

The **FillList AIU** allows for filling a list of field. Each element of the list is a three value record: [FieldName, VariableName, ControlFunction], where FieldName is the text presented to the user for describing the field, VariableName is the name of the variable containing the user input and ControlFunction is an optional validation function that, if possible, is used on the client side to validate the user input (e.g., non empty, only numbers, etc.). The null value signals that the user terminated the AIU through the Quit command, otherwise the AIU returns the list with the filled values.

**USAGE:** the purpose of this AIU is to allow the user to provide the system with an input that is produced by filling one or more fields.

- **SelectChoice**
  
  (ListId, ListDescription, 
  ListOfChoices) 
  :
  (NULL, elementOfListOfListOfChoices)

The **SelectChoice AIU** allows for selecting a tuple (that is returned by the AIU).
The SelectChoice allows for selecting an element among a predefined list of Choices. The null value signals that the user terminated the AIU through the Quit command, otherwise the AIU returns the selected element.

**Usage:** The purpose of this AIU is to allow the user to provide the system with a command or setting the value of a single variable selecting a constant within a predefined list. Roughly speaking it implements a menu or allows for selecting a value for a single variable.

- **SelectMultipleChoice**
  
  (ListId, ListDescription, ListOfChoices)
  
  :{NULL, elementOfListOfChoices}

  The SelectMultipleChoice allows for selecting one or more elements among a predefined list of choices. The null value signals that the user terminated the AIU through the Quit command, otherwise the AIU returns the list with the selected values.

  **Usage:** The purpose of this AIU is to allow the user to provide the system with multiple choices selecting one or more elements within a predefined list. Roughly speaking it implements a non exclusive check box.

### 3.2 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. Some interaction units can also appear in parallel by using the fork construct. This takes into account the common situation in which a single presentation contain more than one AIU at the same time and the case in which we are modelling a task that involves interactions that do not have a pre-defined sequential ordering. (e.g. a form filling and a menu).

The interaction units that return some output (i.e. not void AIUs) can instantiate a variable that can be used in guard conditions to conduct to different states. As the user interact with the system he passes through activity states. Each interaction involves the specification of some data (e.g. an element in a list) and triggers a server computation that bring the user to the next activity.

In order to clarify the use of this model here we provide an example describing how a simple service to select an interesting tourist event and reserve an hotel room can be modelled. We will refer to Figure 1 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, described in Section 5, that allows for modelling the user interaction, generating the interfaces for different devices.

The user starts selecting an interesting events among a list through an InteractTable; once the event has been selected, the user starts inputting data about the city where s/he wants to stay and some details about the period s/he wants to reserve. Since these are two separate tasks they are modelled with two separate AIUs. The city specification is an InteractTable AIU, the details specification is a FillList AIU for which the user is requested to input data about the reservation period. The order of these two task is irrelevant, so they are connected with a fork construct. The final implementation could be a whatever ordering of these tasks or, if the selected device has enough screen space, a single unified view of these two tasks in one single page.

As the user sends input data, the system passes to the next activity. Note that the junction point (the horizontal bar to which the arrows points) indicates that in order to pass to the next activity, the preceding ones must be terminated. The result is modelled as an InteractTable AIU. The result of the query (search for an hotel), in fact, is a set of objects (i.e. hotels), each with a predefined set of attributes. The system, according to devices’ capabilities, could choose to present for each hotel a certain set of attributes while substituting the presentation of the others with a link pointing to further information.

When the user selects a certain hotel, the system shows details about the hotel through a BrowseImage and a BrowseText states. In the "Hotel confirmation" activity, the user is requested for selecting an action to perform between the following list: reserve the hotel or select a new one. This is modelled with a SelectChoice AIU that, depending on the selected device, could be realized in various manners: buttons, links, menu, etc.

According to the selection, the system can proceed to two different activities: return to the "selecting hotel" state or to proceed with the reservation task. If the user decides to proceed with the reservation, the system steps forward to a new fork, below which two concurrent activities take place: the customer data specification and the selection of the kind of payment he wants to use (e.g. credit card payment). Here again, as the case for search parameters, we have concurrent AIUs. This means that they can be implemented in parallel or sequential order. The "Fill_Customer_Data" activity is modelled as a FillList AIU because it is supposed to accept data directly specified by the user. The "Select_Payment_Type" activity is modelled as a SelectChoice AIU because it is supposed to present a predefined list of payment methods from which the user can select the preferred one.

When the user finishes these tasks the system requests for a confirmation and, when the user confirms, the system collects the data and makes the reservation; otherwise the user is routed to the initial state.

This simple example shows how the composition of the abstract interaction units can be done in order to model 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.

4. 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 5. Finally, in this work, even if we devised some AIUs oriented towards image manipulation, we concentrate on textual based AIUs and, consequently, we consider only text/table oriented metrics.

4.1 Devices oriented metrics

Concerning devices we define the following functions:

- int RN(dev) (Row Number), returning the number of rows the device is able to display;
- int CN(dev) (Column Number), returning the number of columns 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 RVs(dev) (Row-based Vertical Scrolling), returning the availability of stepped (i.e., row based) vertical scrolling;
- boolean PVS(dev) (Page-based Vertical Scrolling), returning the availability of stepped (i.e., page based) vertical scrolling;
- boolean CNHS(dev) (CoNtinuous Horizontal Scrolling), returning the availability of a continuous (i.e., pixel based) horizontal scrolling;
- boolean COHS(dev) (COlumn-based Horizontal Scrolling), returning the availability of stepped (i.e., column based) horizontal scrolling;
- boolean PHS(dev) (Page-based Horizontal Scrolling), returning the availability of stepped (i.e., page based) horizontal scrolling;
- boolean WE(dev) (WAP Enabled), true if the device is Wap enabled;
- boolean JE(dev) (Java Enabled), true if the device is Java enabled;
- boolean AA(dev) (Audio Availability), returning the audio channel availability;
in order to display the overall scrolling commands (both vertical and horizontal) needed for a specific device. As an example, we can count the number of characters the AIU needs to be displayed.

4.2 Aius oriented metrics
Concerning the Aius, we investigate some metrics about the text based Aius, distinguishing between table and pure text oriented Aius.

- int CD(dev) (Color Depth), returning the color/black and withe depth (expressed in bit);
- boolean TSA(dev) (Touchable Screen Availability), returning the availability of touchable surfaces.

4.2 Aius oriented metrics

Concerning the Aius, we investigate 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.

The main idea, described in the next session through an example, is to use the above metrics to measure the level of degradation an AIU suffers when implemented on a specific device. As an example, we can count the number of rows and columns the AIU needs to be displayed.

Based on these figures, we can argue that the AIU can be easily displayed for what concerns the number of columns: in fact, it requires at most 40 row based scrolling commands or \(\lceil 40/14 \rceil \) page based vertical scrolling commands and such figures are quite reasonable (form, we define some threshold values). On the other hand, handling 95 columns on a device that is able to display only 30 columns and does not allow for horizontal scrolling is not an easy task. The only way is to present the table to the user in a two steps interaction:

1. The user is presented with a table containing only a subset of the table attributes whose column occupation is less than 30;
2. and additional commands allows for detailing a single row, simulating the horizontal scroll and getting all the events attributes.

The way in which it is possible to implement the above strategy strongly depends on the device computational capabilities (e.g., Java enabled) and on load balancing issues. Here, in order to follow the more robust solution we assume that all the work is performed by the server that, looking at the device capabilities, will produce the needed Java code or Wap pages. As an example, we can see in Figure 2 (a) a possible implementation of the InteractTable AIU on a device with no significative limitations (e.g. a usual browser on a portable PC).

In order to display the same table on reduced capability device we have to reduce the table attributes, producing the table shown in Figure 2 (b) (The image is presented on a Nokia 9500 device, reducing, for test purpose, the number of available rows to 22). In such a table only a attribute subset of the table attributes whose column occupation is less than 30; the device computational capabilities (e.g., Java enabled) and on load balancing issues. Here, in order to follow the more robust solution we assume that all the work is performed by the server that, looking at the device capabilities, will produce the needed Java code or Wap pages. As an example, we can see in Figure 2 (a) a possible implementation of the InteractTable AIU on a device with no significative limitations (e.g. a usual browser on a portable PC).

In order to display the same table on reduced capability device we have to reduce the table attributes, producing the table shown in Figure 2 (b) (The image is presented on a Nokia 9500 device, reducing, for test purpose, the number of available rows to 22). In such a table only a attribute subset of the table attributes whose column occupation is less than 30; the device computational capabilities (e.g., Java enabled) and on load balancing issues. Here, in order to follow the more robust solution we assume that all the work is performed by the server that, looking at the device capabilities, will produce the needed Java code or Wap pages. As an example, we can see in Figure 2 (a) a possible implementation of the InteractTable AIU on a device with no significative limitations (e.g. a usual browser on a portable PC).

The above example provides the feeling on how it is possible to adapt the same AIU to different device. We are currently investigating different implementation strategies and different threshold values to in order 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, implemented through servlets and running under Tomcat.
6. FUTURE WORK AND CONCLUSIONS

In this paper we presented a novel approach for implementing, on a variety of portable devices, simple internet based applications. The main ideas supporting our proposal are:

- a formal model characterizing the user interaction building blocks (AIUs);
- the embedding of such a model within the UML activity diagram;
- the formalization of the characteristics of devices and AIUs through several suitable metrics;
- 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 devices metrics; moreover we are developing a first set of usability studies, to validate our approach.

7. ACKNOWLEDGMENTS

Work supported by the MIUR-FIRB project "MAIS" (Multichannel adaptive Information Systems, http://black.elet.polimi.it/mais/index.php).

8. REFERENCES


