Designing Adaptive Mobile Applications: Abstract Components and Composite Behaviors

Manuel Prieto[2,1] and Miguel A. Sicilia[2]

[1] Telefónica I+D.
C/. Emilio Vargas, 6 – 28043 Madrid, Spain
mjpm@tid.es
[2] Computer Science Department. Carlos III University.
Avda. de la Universidad, 30. – 28911 Leganés, Madrid, Spain
{mprieto, msicilia}@inf.uc3m.es

Abstract. Current commercial software frameworks for the development of mobile applications targeted at heterogeneous devices are based on a paradigm of *abstract* user interface components (or 'controls') that change its rendering depending on device characteristics. In this paper, we approach the problem of extending that paradigm to handle adaptiveness to user models for the purpose of improving usability. A generic approach along with its concrete realization on ASP.NET technology is described. The approach is centered on the notion of componentized adaptive *behaviors*, that can be easily added to user controls by designers. These components can also be chained to come up with more complex behaviors. In addition, some experiences on the automatic, rule-based dynamic addition of such behaviors to concrete controls based on usage data are described.

Keywords. Heterogeneous mobile devices, abstract user interface components, adaptive hypermedia.

1 Introduction

The design of applications for mobile clients poses significant problems derived from two essential constraints [1]. On the one hand, these devices usually provide limited display and interaction capabilities and, on the other hand, their characteristics are widely diverse. Limitations and heterogeneity entail that the tasks that should be analyzed for such systems are different from those found in desktop computers [3], and, as a consequence, the design of the interface must follow different priorities. In this context, adaptiveness becomes an important feature in mobile applications, since it increases navigation efficiency, as has been addressed in previous studies like [6]. In an effort to cope with heterogeneity in limited devices, some industrial user interface development frameworks have been crafted recently. Most notably, Microsoft's Mobile Internet Toolkit (MIT)[3] and Java Micro Edition[4] provide a programming paradigm based on what may

[3] http://www.gotdotnet.com/team/mit/
[4] http://java.sun.com/j2me/

be called 'abstract user interface components' (AUIC), which in essence are common interface components (e.g. lists, labels) that are prepared to be rendered — and even reshaped — in different ways depending on the characteristics of the device interacting with them. In this paper, a model for the development of adaptive interfaces for heterogeneous and limited devices is described, explicitly targeted towards AUIC-based frameworks. A concrete instantiation of the model on the MIT framework is also described — both the model and the implementation are based on previous work described in [4]. The most salient feature of our approach is that adaptiveness is associated to abstract components, easing the design of personalized interfaces by means of the possibility of selecting and arranging some generic adaptive behaviors for specific component instances. In addition, an initial exploration on the automatic selection of adaptive behaviors based on usage data has been approached. The rest of this paper is structured as follows. In Section 2, the principal elements of our model are described in abstract terms. Section 3 sketches the concrete realization of the model on top of the MIT framework, along with a brief description of some concrete automatic adaptation functionality. Finally, conclusions and future research directions are provided in Section 4.

2 A Model of Composable Adaptive Behaviors

An abstract model of AUIC for personalized mobile applications must provide modelling constructs for describing devices, interface components and users. In addition, it must provide a paradigm for expressing adaptations. Devices can be described by device profiles in the form $\wp_x = \{(p_i, v_i) | p_i \in P \wedge v_i \in type(p_i)\}$, where P denotes a set of device parameters (capabilities) defined in a pre-existing ontology — as the FIPA device ontology[5]— and $type(p)$ denotes the type of the parameter (e.g. integer, string, set, sequence). Parameters may describe software (e.g. operating system), hardware (e.g. memory, screen height, resolution) or network (e.g. quality of service) characteristics. A generic attribute-based user model can be specified by considering in a similar way a set $U = \{u_m\}$ where $al(u_j)$ denotes the collection of attribute–value pairs describing the concrete user. Our model of the structure of the application is limited to describing the AUIC components in set C that contains, in the form $\mathcal{A} = \{c_k | c_k \in t_x\}$, where $t_x \in \mathcal{T}$ denotes an specific *component type* in the set of types $\mathcal{T}$. As usual in component frameworks, each component type is described by a set of predefined attributes denoted by $at(t_x)$. Types and subtypes may be interrelated in a generalization–specialization hierarchy so that if type b specializes type a, denoted by $a \succ b$, then $at(a) \subset at(b)$. Following the above definitions, our AUIC–centric paradigm describes adaptation in terms of an ordered sequence of behaviors attached to each specific component:

$$b(c) = b_1, \ldots b_l, \;\; l \geq 0, \; c \in t, \; t_i \in \mathcal{T} \tag{1}$$

[5] FIPA Device Ontology Specification, doc. number XC00091C, available at http://www.fipa.org.

Each b_i in expression (1) represents a software entity that transforms the state of component c by changing some of the values of the attributes in $at(t)$. Behaviors are also typed, so that each b_i has a type $h \in \mathcal{B}$. In addition, each type of behavior h is associated to exactly one type of component in $c(h) \in \mathcal{T}$. By virtue of this association, behaviors of type h can only be applied to components of that type. Composability can be easily introduced as simple rewriting, so that a sequence of behaviors $b_r \ldots b_s$ can be labelled as b_q, so that the new label functions as a higher-level, more complex behavior that sequentially applies its contained elements. It should also be noted that order is important in (1) since the order of processing b_i, b_j need not produce the same final result than the order b_j, b_i. In general, behaviors produce a change in its associated control dependant on device characteristics and the characteristics of the current user (or characteristics of one of the groups to which he/she belongs), which can be denoted abstractly as $b : 2^U \times C \times \wp \rightarrow C'$. All the definitions given above can be easily recognized in current commercial mobile development frameworks, and in consequence, they provide a basis for extending them in a straightforward way, as will be demonstrated in what follows.

3 MIT Implementation of Adaptive Behaviors

Microsoft's Mobile Internet Toolkit (MIT) provides a server runtime that allows the automatic adaptive presentation of Web contents to a number of supported devices. Adaptation is based on a set of mobile server controls, which are an abstraction of the rendering and interaction features of specific languages like WML and cHTML (the approach is similar to that of [2] although their scope is different). In our extension of the MIT, application designers can add personalization behaviors by simply typing it in the control's definition. For example, the following code fragment shows the declaration of an abstract component ($p \in C$, with $b(p) = f_2, f_5$) that specializes a list, called `PersonalizedList`. Note that two behaviors `f5` and `f2` are specified in one of the attributes.

```
<mobile:Form id="Form1" runat="server"> <shadow:PersonalizedList
id="p" runat="server" NAME="p_1" behaviours="f2;f5">
    <Item Text="Palacio de la Prensa" Value="http://www.pmc.es/" />
    <Item Text="Benlliure" Value="http://www.benlliurecines.es/" />
       ...
</shadow:PersonalizedList> </mobile:Form>
```

Behaviors (in $\mathcal{T}$) are defined in a separate XML file with the following appearance:

```
 <behaviors-def> <behavior>
   <name>f1</name>   <desc> Discard visited links. </desc>
   <class>MobilePrb1.DeleteVisitedBehavior</class>
   <file>C:\\Inetpub\\wwwroot\\M1\\MobilePrb1.dll</file>
   <type>PersonalizedList</type>
</behavior> <behavior>   <name>f2</name>  ...
```

Each behavior definition contains its name and description, the class that realizes the personalization, the physical file in which the code is located and the abstract control class it can adapt. This programming model also allows to customize the behaviors without recompilation. Customized mobile server controls can be defined by simply subclassing them. For example, the component called `System.Web.UI.MobileControls.List` can be subclassed to carry out the adaptation of lists. The following C# code fragment illustrates it.

```
public class PersonalizedList: System.Web.UI.MobileControls.List {
private void Page_Load(object sender, System.EventArgs e) {
  GeneralPage page = (GeneralPage) this.Page;
  ListPersonalization persEngine = new ListPersonalization();
  MobileListItemCollection result =
        persEngine.getPersonalizedList( this.Items,  page.userId,
                          page.Request.Path, behaviours,  restr_disp);
  while (this.Items.Count > 0){ this.Items.RemoveAt(0); }
  foreach (MobileListItem it in result){
    it.Value="ListRedirect.aspx?dest=" + it.Value +
           "&page=" + page.Request.Path + "&user=" + page.userId;
        this.Items.Add(it);
   }
    this.ItemsAsLinks = true; }
```

Basically, the `ListPersonalization` class encapsulates the behaviors that depend on the model of the user, which is identified by the `userId` attribute. The collection of behaviours is extracted from the declaration of the control (like the one showed before). The collection called `restr_disp` of device capabilities is obtained from the run-time information provided by ASP.NET. Note also that the items in the list are changed to ListRedirect.aspx so that the navigation of the user is recorded. Behaviors can be aggregated according to the *Composite Filter* pattern [7]. The pattern is embedded in the call to `getPersonalizedList`, and the reflective capabilities of the .NET framework are used to dynamically instantiate and invoke the filters.

```
public MobileListItemCollection getPersonalizedList (
           MobileListItemCollection items, string userid, string page,
           string behaviours, MobileCapabilities capabilities)
{  ArrayList b = behaviorsParser(behaviours);
   foreach (string be in b) {
       items = doFilter(items, userid, page, be, capabilities);}
   return items; }
```

Access to the user model server (or a wrapper to existing ones) is implemented as a standard-based Web Service, allowing for the federation of user model servers (details are not covered here).

Behaviors can be selected by designers to improve user experience according to their knowledge of the context of each specific control instance. But in some cases, the system may be able to automatically set (or rearrange) behaviors. Currently, we have formalized some of these updates through rules of two types: (a) oriented towards performance, e.g. given that o is a sorting behavior and d is

a behavior that removes items in a list: $\{o_i, d_j\} \subseteq b(c) \Rightarrow i \geq j$ to avoid unnecessary sorting processing time, and (b) based on usage patterns. The second category aims at direct improvements of usability and require heuristic approaches. For example, some lists show highly volatile information (like soccer scores), so that once consulted by the user, they're rarely re-visited, and they also change frequently. This usage pattern may be detected whenever periodical variations are detected, and most users do not consult the list more than once, resulting in chaining a 'remove once visited' behavior to the list. Sorting behaviors are another case of automatic behavior chaining. Given a static list control $l \in C$, sorting can be activated if the following rule holds: $\exists l.item_i,\ most\ visits(l.item_i)$ or $\exists l.item_i, l.item_j,\ visits(l.item_i)\ mgt\ visits(l.item_j)$ signifying that there exist items that receive most of the visits or there are large differences in usage for different items. Quantifier *most* and the expression 'much greater than' (mgt) are implemented as described in [5] through fuzzy sets.

4 Conclusions and Future Work

A component-based model has been described for extending AUIC frameworks with control-oriented adaptation. The model can be easily implemented in existing commercial frameworks, and a case study of a MIT-based implementation has been sketched. Future work should deal with the automatic selection of behaviors and also with adaptations involving more than one control.

References

1. Billsus, D., Brunk, C.A., Evans, C., Gladish, B and Pazzani, M: Adaptive Interfaces for Ubiquitous Web Access. Communications of the ACM 45(5) (2002):34–38
2. Gaedke, M., Segor, C. Gellersen, H.W.: WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. In: Proceedings of the 2000 ACM Symposium on Applied Computing (SAC 2000), (2000):19-21
3. Landay, J.A. and Kaufmann, T.R.: User Interface Issues in Mobile Computing. In: Proceedings of the 4th Workshop on Workstation Operating Systems (1993):40–47
4. Prieto, M. and Sicilia, M.A.: Designing Agent-Based Personalized Filtering Behaviors for Heterogeneous Mobile Internet Devices. In Proceedings of the 1st International Workshop on Practical Applications of Agents and Multiagent Systems (2002):125–134
5. Sicilia, M.A., Díaz, P., Aedo, I. and García, E.: Fuzzy Linguistic Summaries in Rule-Based Adaptive Hypermedia Systems. In: Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems (2002):317–327
6. Smyth, B. and Cotter, P.: Personalized Adaptive Navigation for Mobile Portals. In: Proceedings of the 15th European Conference on Artificial Intelligence Prestigious Applications of Intelligent Systems (2002)
7. Yacoub, S.M.: A Versatile Filter Pattern. In: Proceedings of the EuroPLoP 2001 Conference, Irsee, Germany, 4-8 July (2001)