A user-centric approach to adaptable mobile interfaces

Aitor Almeida, Pablo Orduña, Eduardo
Castillejo, Diego López-de-Ipiña
DeustoTech –Deusto Institute of Technology
Avenida de las Universidades 24,
48007 Bilbao (Spain)
{aitor.almeida, pablo.orduna, eduardo.castillejo,
dipina}@deusto.es

Marcos Sacristán
Treelogic
Parque Tecnológico de Asturias · Parcela 30
E33428 Llanera · ASTURIAS (Spain)
marcos.sacristan@treelogic.com

Abstract

The adoption of ICT assistive technologies by elderly and disable collectives has been slow and unsatisfactory. However, a bigger adoption could contribute to their independent living significantly. As the average age of the population increases in the most developed countries, this becomes an unavoidable problem. To tackle this problem, this paper devises a framework that facilitates the process of creating interfaces that adapt themselves to the specific capabilities of each user. Furthermore a Fuzzy Knowledge-Eliciting Reasoner is proposed that infers new capabilities from the existing ones. Using this reasoning engine, developers can use more natural concepts when stating the code adaptation directives.

1. Introduction

There are two user collectives which experience most difficulties when accessing technology, although they could benefit highly from their adoption: elders and people with disabilities. Studies [1] show that there are less than half computer users among people with disabilities that among non-disabled people. This same issue is encountered among persons aged 65 and above and the rest of the users. And the gap broadens even more when we start comparing elderly people with and without disabilities. With the ageing of Europe [2] this neglected user base is going to become even more important in the near future. This situation is a frustrating contradiction. Computers, mobile devices and Internet present an incredible opportunity to increase the independence of these collectives. This is why interfaces that adapt themselves to the user's capabilities are a must in AAL [3]. Additionally, the emerging Future Internet paradigm, aware of this fact, proposes as a key pillar of it, Internet for

and by the People, where services and contents are adapted to the needs, capabilities, social and cultural background of people

Against this background we have developed a framework that facilitates the process of creating adaptable user interfaces. This framework makes possible to use Preprocessor Directives to state which code regions are suitable for different user and device capabilities. Furthermore we have developed a Fuzzy Knowledge-Eliciting Reasoner that can infer new capabilities from the existing ones and that uses trend data to compare and put into context those capabilities. Thanks to this fuzzy reasoner developers can employ more natural concepts in the Preprocessor Directives, making simpler their use.

The remainder of the paper will consist on the analysis of the related work on Section 2, a description of the system architecture and the user and device capabilities in Section 3, an explanation of the use of the Preprocessor Directives in Section 4, the description of the Fuzzy Knowledge-Eliciting Reasoner in Section 5, an explanation of the developing cycle in Section 6 and finally the conclusions and future work on Section 7.

2. Related Work

User-centric adaptation has already been used in other areas. In [4] and [5] authors develop a middleware to adapt services to context changes based on a user-centric approach, authors in [6] use emotion sensing to adapt the news to the user mood.

Several frameworks have used different techniques to provide an easy to use and powerful tool to create adaptable user interfaces. We have identified three main approaches to this problem: Custom mark-up languages (as those used by OpenLaszlo [8]), use of factories (like Google Web Toolkit [9] and EMI2lets [7]) and the

Preprocessor Directives (as used in Antenna [10] and J2ME Polish [11]). All these alternatives have their own strengths and weaknesses (see Table 1)

Approach	Advantages	Disadvantages
Custom mark-up language	-The code is independent from the target platform.	-The developer must learn a new language. -The executable code creation must be implemented once for each target platform.
Factories	-Easier to integrate with an IDE. -Interfaces can be dynamically adapted in execution.	-Strongly coupled with the programming language. -The library must be implemented once for each target platform.
Preprocessor directives	-The code is independent from the target platform. -The developer does not have to learn a new language, only some directives.	-Once preprocessed interfaces cannot be changed in execution.

Table 1. Comparison of the different approaches

After analyzing the alternatives we have concluded that the approach used by Antenna and J2ME Polish is the most fitting one for our requirements. Decoupling the framework from the programming language provides a versatility not attainable with the other approaches, making the framework suitable to be used in the development of any application. What is more, because there is no need to learn a new language any developer can easily include the necessary directives in his code.

Our framework provides two important innovations over the functionality of J2ME Polish and Antenna. First, it does not only take into account the capabilities of the devices when processing the preprocessor directives in order to adapt the interface. It also considers the user capabilities (sensorial, cognitive and physical) to create the most suitable user interface. We consider that the most important element in an application is the user, and so it must be the application the one to adapt to the user's abilities.

Secondly, we provide a fuzzy-reasoning mechanism to infer new user and device capabilities from those specified in the profiles. This allows the creation of richer user and device profiles and a more personalized interface. It also makes possible the use of fuzzy concepts in the Preprocessor Directives (e.g. *the user sees well, the device screen is big*), making them easier to use by the developers.

3. System Architecture

In the following section we describe the architecture of the developed framework. This framework is part of the PIRAmIDE [13] project, that has as one of its main objectives to analyze and develop the capabilities of the mobile devices as tools for the interaction with the everyday objects.

Our framework is divided in two main elements, the *Mobile Client* and the *Application Server*. The *Mobile Client* manages the user and device profiles and the application search and download processes. The device profile is automatically generated using the model characteristics, but the user has to manually create his profile (which states his capabilities).

The *Application Server* is in charge of the adaptation process and can be divided in 4 sub-elements:

- *Application Repository*: This repository stores all the available applications of the system. Users can perform searches over the repository and see if the application that they want is available for their platform.
- *Fuzzy Knowledge-Eliciting Reasoner*: Infers new capabilities using the existing ones. This reasoning engine is explained in Section 5.
- *Preprocessor*: Adapts the code to the user and device capabilities using the *Preprocessor Directives* contained in the source code. This process creates a fully personalized application. The *Preprocessor* uses the *Fuzzy Knowledge-Eliciting Reasoner* to infer new user and device capabilities, making explicit the implicit knowledge in the user and device profiles.
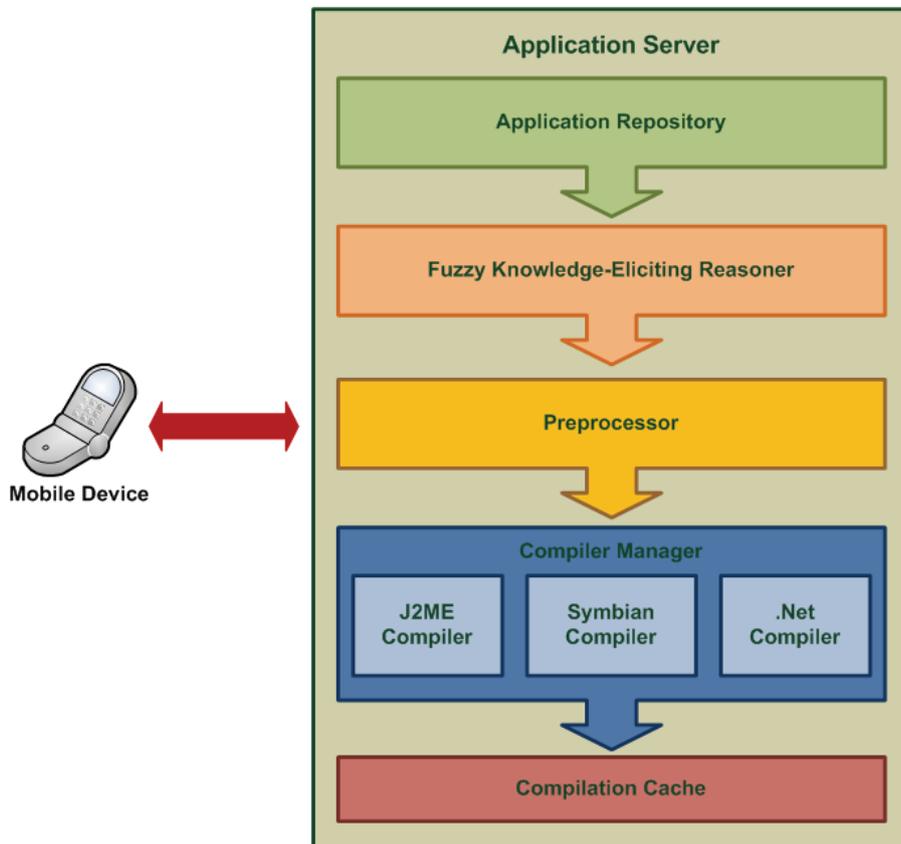
Figure 1. System architecture

- *Compiler Manager*: Selects the correct compiler for the target platform. It also compiles and builds application using the configuration specified by the developer.
- *Compilation Cache*: This element stores a cache of compiled applications along their compilation configuration to avoid to continually process the most popular ones. Each compilation is identified by three elements, the values of user and device capabilities used for the preprocessing, the identifier of the program and the target platform.

The typical lifecycle of an application is the following one:

1. The developer creates a new application using the Preprocessor Directives defined by our framework. The developer is also in charge of creating a file that has to specify the target platforms (Symbian, Android, .NET…) of the application and its compilation configuration.
2. The application is uploaded to the Application Repository. This repository stores all the instances of the same application. One application usually has different versions (v1.1, v1.2…) and different target platforms
3. The user searches the repository for an application and sends an install request along his user and device profiles.

4. The selection of the code for the compilation will be determined by the user and device characteristics. E.g. if the user is colorblind and a specific treatment is applied in the code for colorblindness then this code region will be selected. Then the Application Server will select those versions that have a suitable target platform.

5. If a compilation exists in the Compilation Cache with the same Compilation Configuration (composed by the application name, the application version, the target platform and the relevant user and device characteristics) this compilation will be sent to the user. Otherwise we process the user and device capabilities to infer new ones using the Fuzzy Knowledge Eliciting Reasoner. The Preprocessor uses the capabilities and the annotated source code to create a fully customized code suitable to the user capabilities.

6. The new compilation is stored in the Compilation Cache and the user receives the binary of the selected application.

3.1. User and device capabilities

Initially, the user capabilities were divided into four groups:

- *Sensorial:* Those capabilities related to the five senses.
- *Cognitive:* Capabilities related to memory, attention span and reasoning ability.
- *Physical:* In this category fall those abilities related to the mobility of the user.
- *Communicational:* Those capabilities related to the communicational abilities of the user.

Later, we added another group, the combined capabilities. This group takes into account the synergies generated by the accumulation of various disabilities (this occurrence is very usual among elderly people).
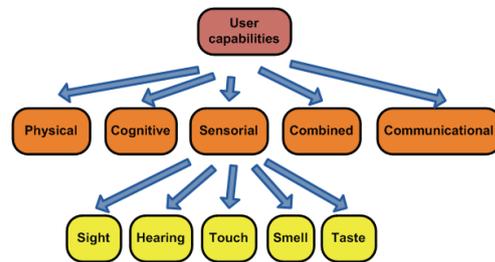

Figure 2 Partial representation of the taxonomy of user capabilities

On the other hand, we have used WURFL 2.9.5 [14] to identify the device capabilities. Some of the display capabilities for the Nokia N97 can be seen in Figure 3

```
<group id="display">
    <capability name="columns" value="17"/>
    <capability name="dual_orientation" value="true"/>
    <capability name="max_image_width" value="360"/>
    <capability name="rows" value="13"/>
    <capability name="resolution_width" value="360"/>
    <capability name="resolution_height" value="640"/>
    <capability name="max_image_height" value="640"/>
</group>
```

Figure 3 Display capabilities for the Nokia N97

4. Preprocessor directives

The preprocessor identifies the directives when they start by *//#* in languages that support inline comments starting by *//*, such as Java, C# or C++, *#//* in languages that support inline comments starting by *#*, such as Python or Perl, and '*//* in VB.NET. There are three types of directives:

4.1. Conditionals

The preprocessor can avoid the compilation of fragments of code if certain conditions are matched. These conditions can include calls to functions provided by the system. Basic string and math functions are available, including *lowercase*, *trim*, *contains*, *round* or *sqrt*, as well as functions to check if a certain variable is available. The conditions can be embedded, as shown in the example below:

```
//#if defined(${piramide.devices.height})
    //#if ${piramide.devices.height} > 150 * 2
            addTenRows();
    //#elif ${piramide.devices.width} > 200
            addTenCols();
    //#else
            addFiveRowsAndCols();
    //#endif
//#else
        addFiveRowsAndCols();
//#endif
```

Figure 4 Sample of "if..elif..else..endif" directives in Java, C# or C++

The syntax of the conditions follows the syntax used by the Python programming language.

4.2. Error handling

Under certain situations, it is useful to report an error in order to manage unhandled situations. For instance, an application developer might provide a video interface for a regular user and a label for users with earring impairment, but not provide any alternative if the user has visual disability. In this case, the developer can explicitly force the system to fail, providing a human readable message.

In the code below, the developer reports that the application could not be compiled due to configuration issues:

```
'//if not defined(${piramide.devices.video})
'//error Video variable must be configured
'//endif
```

Figure 5 Sample of error directive in VB or VB.NET

4.3. Parameterizing code

Whenever it is required, the developer can directly store in programming variables the system ones. This way, the developer can adjust programmatically to the exact values of the variables. As an example, the developer of the code below will adapt a widget to the exact screen size:

```
screen_height = 0
#// screen_height = ${piramide.devices.height}
window.height = screen_height
```

Figure 6 Sample of hidden directive in Python

However, two users with similar mobile devices that have a slightly different value in one variable will require two compilations if this variable is bound to a Java variable using the hidden directive. Therefore, if the programmer only uses the variable to check if the screen size is bigger than a certain quantity, then it will require several compilations that will have the same exact functionality for different devices with a bigger screen. The cache manager will not be able to handle this situation because the actual screen size will be present even in the compiled code, so any test to check if two compilations are equal will fail. So, since it is very easy to misuse this directive, it is in general discouraged although still supported for certain situations that otherwise could not be implemented.

4.4. Security

The backend interpreter used is Jython [12], an open source Python implementation developed in Java. The conditionals variables are directly executed. To avoid malicious code, the system checks that all the names of functions called are in a white list of the functions provided. This way, a malicious user can't import modules or call system functions that could have an impact on the system. The malicious user can still create big variables that generate heap overflow errors, but this only affects to the current compilation, reporting an error to the user.

5. The Fuzzy Knowledge-Eliciting Reasoner

There are situations where the capabilities specified in the user and device profiles are not suitable to be used directly. For example, the developer may want to show certain video only if the screen of the device is "big". The main problem with this scenario is that the concept "big" is not directly related to one value and is a relative value (which implies that what is a big screen today probably won't be big in 2 years). For this reason we have developed the Fuzzy Knowledge-Eliciting Reasoner.

The goal of the Fuzzy Knowledge-Eliciting Reasoner (see Figure 1) is to identify new capabilities using the already existing ones and to fuzzyfy them. To do this we have defined a set of

fuzzy rules that take as input numeric values from the existing capabilities and create symbolic values for the new ones. An example for the reasoning that takes place in this stage would be: "*If the user has a lot of dioptres and the screen size is normal the visualization problem is high*" or "*If the resolution is big and the screen size is big the video suitability is very high*". This reasoning will be modeled with fuzzy rules (see Algorithm 1).

```
IF  screensize  IS  big  AND
resolution IS normal
 THEN video IS high;

IF  screensize  IS  big  AND
resolution IS big
 THEN video IS very_high;
```

Algorithm 1.   Fuzzy rules for the video suitability of the device

The main problem we have encountered using fuzzy rules is that we need to fuzzify the crisp variables encountered in the databases (in our case WURFL 2.9.5[14]). This raises some challenging questions. What do we consider a "big" screen size? How can we identify what characteristics are inherent of the average mobile device? These concepts are relative to the values of other device models. One screen is big if its height and width are larger than the average values of the other models. But all the device models can not have the same weight in the calculation, not all the device models have sold the same number of units. This is why the most popular models should have more weight during this calculation.

In order to calculate the popularity of one device we have to adjust it with its "age". Popularity fades with the passing of time. Users tend to change their mobile phones frequently, drastically altering the perception of what is a big screen from one year to another. Our proposed solution uses Google Trends [15] to identify the popularity of each mobile device. Mobile phone plans have an average duration of 15 and 24 months. Bearing this in mind we use the trend information of the last three years (2007, 2008, 2009 and the first 2 months of 2010). In the next versions of the algorithm we plan to use a decay measure to take into account the degradation of the popularity of a device over time.

While this number does not represent the sale volume, we think that it is a good indicator of the interest shown by the consumers in a specific model. Due to the lack of data regarding the real sale volume for most mobile devices is one of the few available indicators. This trend value can change drastically from one location to another, the most popular devices are not the same in Japan and Spain. To tackle this problem we support the geolocation of the results to filter them according to the needs of the developers.

According to Google Trends, the most popular mobile devices worldwide it this period of time have been (ordered by popularity): Nokia 6300, Blackberry Storm, Apple iPhone, Nokia N97. Blackberry Pearl, Nokia e71, LG Viewty, Nokia N73, Nokia e65, Nintendo DS.

Once the popularity values for all the devices in the WURFL database have been retrieved, the membership functions of the different crisp capabilities are calculated. For example, in Figure 7 can be seen the adjusted values of popularity expressed in number of searches performed (vertical axis) for the values of the screen resolution represented as the result of multiplying the height and width (horizontal axis). As can be seen in the graph the majority of the population is gathered between 50000 and 100000.
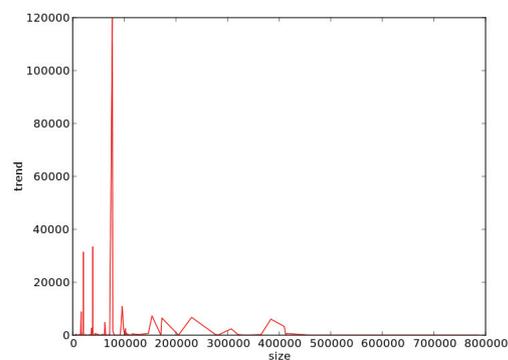

Figure 7 Popularity calculation for all the resolutions of the mobile devices in WURFL

We use this popularity value to calculate the membership function of each variable (see Figure 8). In this case the screen resolution has 3 linguistic terms: small (dashed), normal (solid) and big (dotted).
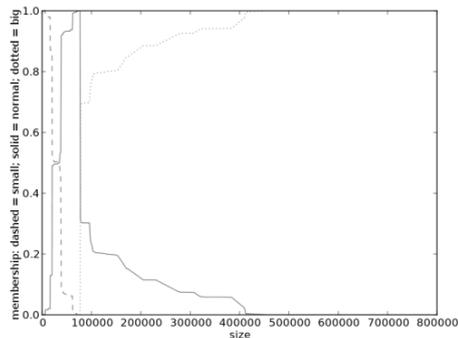
Figure 8. Membership function for the resolution of the screen using 3 linguistic terms: small (dashed), normal (solid) and big (dotted).

Finally we use these membership functions in the fuzzy rules described previously in this section.

6. The developing cycle

When creating an application using our framework the developer must take the following steps:

1. If the developer wants to use the fuzzy logic capabilities of the framework he must define the rules used by the Fuzzy Knowledge-Eliciting Reasoner. The membership functions will be calculated automatically by the framework.
2. Using the base capabilities and the new ones defined in the previous step the developer programs the application annotating the code with the Preprocessor Directives. Using a local instance of the server he can compile the application with different configurations to check the results.
3. Once the application is finished the developer creates a Maven file with the configuration to compile and build the application
4. Finally the developer uploads the application source code, the fuzzy rules and the Maven file to the server to make the application available.

7. Conclusions

Elderly people use technology reluctantly, this is why it is even more important for the user interfaces to be easy to use in AAL applications. The creation of applications adapted to the user's capabilities should be one of the most important tasks in AAL, but in many cases we are too focused on the technology and forgot people.

Against this background, this paper has presented a framework that enables the creation of applications suited for different user and device capabilities easily. We have also discussed a fuzzy-logic based inference mechanism that allows identifying new capabilities based on those ones already known. This inference mechanism also permits to use fuzzy concepts on the creation of the Preprocessor Directives, enabling the developers to abstract from the crisp values (*the user has less than 3 dioptres*) in favor of more natural concepts (*the user can see without a significant problem*).

Future work will focus on making the framework more developer friendly with the addition of a wizard that will assist developers in the process of rule creation and code annotation. We also plan to integrate the framework with an IDE. We will also use additional device databases to improve the devices' data. Finally we will test the created interfaces with users with different capabilities to measure their usability. To this end we have already established contacts with several associations. We expect that the developed framework will be available as open source before the end of 2010.

References

[1] Kaye, H.S., Computer and Internet Use among People with Disabilities, in Disability Statistics Report 2000, Department of Education, National Institute of: Washington D.C.

[2] A. Walker. Ageing in Europe – challenges and consequences. Zeitschrift für Gerontologie und Geriatrie, Springer Berlin / Heidelberg. Volume 32, Number 6, 1999.

[3] Ambient Assisted Living Joint Programme, http://www.aal-europe.eu, (2010)

[4] Hyo-In Ahn , Jae-Eun Lee , Yong-Ik Yoon. A Middleware for User Centric Adaptation Based on Fuzzy in Ubiquitous Environment. Sixth International Conference on Advanced Language Processing and Web Information Technology, 2007.

[5] M. Abe, Y. Morinishi, A. Maeda, M. Aoki and H. Inagaki. Coool: A life log collector integrated with a remote-controller for enabling user centric services. Digest of Technical Papers International Conference on Consumer Electronics, 2009.

[6] S. Mostafa Al Masum , H.t Prendinger , M. Ishizuka. Emotion Sensitive News Agent: An Approach Towards User Centric Emotion Sensing from the News. IEEE/WIC/ACM International Conference on Web Intelligence, 2007.

[7] D. López de Ipiña, I. Vázquez, D. García, J. Fernández, I. García, D. Sainz and A. Almeida. A Middleware for the Deployment of Ambient Intelligent Spaces. Ambient Intelligence in Everyday Life, Springer, Lecture Notes in Artificial Intelligence, LNAI 3864, ISSN 0302-9743, pp. 239-255, 2006

[8] OpenLaszlo. www.openlaszlo.org (2010)

[9] Google Web Toolkit. code.google.com/webtoolkit (2010)

[10] Antenna. antenna.sourceforge.net (2010)

[11] J2ME Polish. www.j2mepolish.org (2010)

[12] Jython.. www.jython.org (2010)

[13] PIRAmIDE Project. www.piramidepse.com (2010)

[14] WURFL 2.9.5. wurfl.sourceforge.net (2010)

[15] Google Trends. www.google.com/trends (2010)