Pers Ubiquit Comput (2011) 15:419–429
DOI 10.1007/s00779-010-0359-8

ORIGINAL ARTICLE

Imhotep: an approach to user and device conscious mobile applications

Aitor Almeida · Pablo Orduña · Eduardo Castillejo · Diego López-de-Ipiña · Marcos Sacristán

Received: 29 September 2010 / Accepted: 4 December 2010 / Published online: 19 January 2011

Abstract As the dependence on mobile devices increases, the need for supporting a wider range of users and devices becomes crucial. Elders and people with disabilities adopt new technologies reluctantly, a tendency caused by the lack of adaptation of these technologies to their needs. To address this challenge, this paper describes a framework, Imhotep, whose aim is to aid developers in the accessible application creation process, making the creation of user-centered applications easier and faster. Our framework allows to easily adapt the applications to the constraints imposed by the user capabilities (sensorial, cognitive, and physical capabilities) and device capabilities by providing a repository that will manage the compilation and deployment of applications that include a set of preprocessor directives in the source code. These directives are enhanced with concepts that are automatically adjusted to the current trends of mobile devices by using a Fuzzy Knowledge-Eliciting Reasoner. Our final goal is to increase the number of applications targeted to elders and people with disabilities providing tools that facilitate their development. The paper also describes the evaluation of both the accuracy of the fuzzy terms generated for mobile devices and the usability of the proposed platform.

Keywords User interfaces · Accessibility · Adaptable interfaces · Fuzzy logic · User aware

1 Introduction

Technologies in general and mobile devices in particular are great tools to increase the independence of their users. But the two user collectives that can benefit the most from their use are precisely those experiencing most difficulties when accessing them: 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 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. In order to address these issues, we have developed Imhotep, 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

A. Almeida (✉) · P. Orduña · E. Castillejo · D. López-de-Ipiña
DeustoTech - Deusto Institute of Technology,
University of Deusto, Avda Universidades 24,
48007 Bilbao, Spain
e-mail: aitor.almeida@deusto.es

P. Orduña
e-mail: pablo.orduna@deusto.es

E. Castillejo
e-mail: eduardo.castillejo@deusto.es

D. López-de-Ipiña
e-mail: dipina@deusto.es

M. Sacristán
Treelogic, Parque Tecnológico de Asturias,
Parcela 30, 33428 Llanera, Spain
e-mail: marcos.sacristan@treelogic.com

Table 1 Comparison of the different approaches

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

have developed a Fuzzy Knowledge-Eliciting Reasoner that can infer more abstract and richer capabilities from the base capabilities defined by the user 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 Sect. 2, a description of the system architecture and the user and device capabilities in Sect. 3, an explanation of the use of the Preprocessor Directives in Sect. 4, the description of the Fuzzy Knowledge-Eliciting Reasoner in Sect. 5, the evaluation of the framework in Sect. 6, the presentation of an use case in Sect. 7 and finally the conclusions and future work on Sect. 8.

2 Related work

User-centric adaptation has already been used in other areas. In [4, 5], authors developed a middleware to adapt services to context changes based on a user-centric approach, authors in [6] used emotion sensing to adapt the news to the user mood. In [7], authors describe a mechanism to adapt the user interface to some physical characteristics of the user like his height or the distance to the screen, without a special emphasis to the disabilities. In [8], the authors defined a context model for information adaptation that takes into account the user needs and preferences.

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[1]), use of factories (like Google Web Toolkit[2] and EMI2lets [9]) and the Preprocessor Directives (as used in Antenna[3] and J2ME Polish[4]). All these alternatives have their own strengths and weaknesses (as described in Table 1).

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 capabilities) 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 that adapts itself to the user's abilities and not vice versa.

Second, 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

This section describes the architecture of the developed framework. The Imhotep framework is part of the PIRAmIDE[5] project, which 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.

[1] OpenLaszlo. http://www.openlaszlo.org

[2] Google Web Toolkit. code.google.com/webtoolkit

[3] Antenna. antenna.sourceforge.net

[4] J2ME Polish. http://www.j2mepolish.org

[5] PIRAmIDE: Personalizable Interactions with Resources on AMI-Enabled Mobile Dynamic Environments. http://www.piramidepse.com

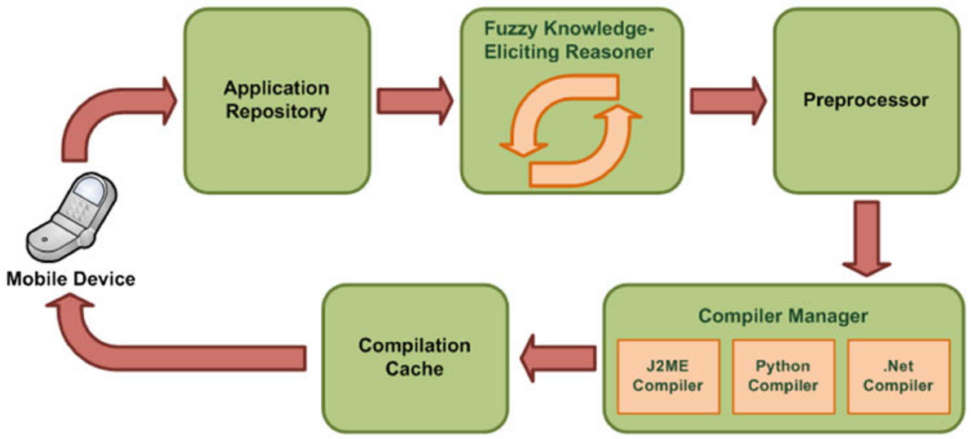


Fig. 1 System architecture

Our framework (see Fig. 1) is divided into two main elements, the Mobile Application Manager and the Application Server. The Mobile Application Manager 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 into 5 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 user and device capabilities, making explicit the implicit knowledge in the user and device profiles. This reasoning engine is explained in Sect. 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. This process is explained in Sect. 4
– *Compiler Manager:* Selects the correct compiler for the target platform. It also compiles and builds the 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 processing the most popular ones. Each compilation is identified by three elements, the identifier of the program, the values of user and device capabilities used for the preprocessing and the target platform.

The typical life cycle 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 compilation configuration 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 profile and device identifier.
4. The selection of the code for the compilation will be determined by the user and device characteristics. E.g. if the user is color-blind and a specific treatment is applied in the code for color-blindness, 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

We have created two extensive taxonomies for the 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).

On the other hand, we have used WURFL (Wireless Universal Resource File) 2.9.5[6] to identify the device capabilities. WURFL is an open source XML database containing the data about the features and capabilities of a large number of mobile devices. Capabilities are classified into different groups, some of the most useful ones for the developers using Imhotep are:

- *product_info:* This group contains information about the model and brand name, the operating system, if the device is wireless, the pointing method used (joystick, stylus, touch screen, click wheel...), if the device has a qwerty keyboard, etc.
- *display:* Contains information about the device screen like the resolution, number of columns and rows, dual orientation, etc.
- *image_format:* Supported image formats.
- *bearer:* radio technologies supported by the device.
- *playback:* Codec support for the mobile device.
- *streaming:* Streaming capabilities of the mobile device.
- *sound_format:* Sound formats supported by the device.

4 Preprocessor directives

The code provided by the developer will be annotated with a set of directives that the preprocessor will compute. These directives define how the final source code must be generated, providing conditions for certain regions of code to be added or skipped, and adding Imhotep variables that the preprocessor will adapt for each compilation. The preprocessor therefore will parse the original annotated source code and will generate the code that will actually be compiled.

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.

[6] WURFL (Wireless Universal Resource File). wurfl.sourceforge.net

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 Fig. 2.

The syntax of the conditions is based on 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 Fig. 3, the developer reports that the application could not be compiled due to configuration issues:

4.3 Parameterizing code

Whenever it is required, the developer can directly store in programming variables the system ones. In this way, the developer can adjust programmatically to the exact values of the variables. As an example, the developer of the code presented in Fig. 4 will adapt a widget to the exact screen size.

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 whether the screen size is bigger than a certain quantity, then it will require several compilations that will have the same exact functionality for

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

Fig. 2 Sample of "if…elif…else…endif" directives in Java, C# or C++

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

Fig. 3 Sample of error directive in VB or VB.NET

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

Fig. 4 Sample of hidden directive in Python

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 whether 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, 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 cannot 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

Among the design objectives of this framework, one is to help programmers without expert knowledge in accessibility to create accessible interfaces, thus making the development process more friendly and accessible. One problem we identified during the development of the framework was that these non-expert developers usually do not have the knowledge to identify the exact values to be used in the preprocessor directives. For example, a developer might want to know whether a processor is "fast" or a screen is "big", without having to deal with specific values. Working with values closer to natural language rather than crisp values eases the use of preprocessor directives. We encountered several problems with this approach:

– Obviously, one screen is big when we compare it with the other screens in the market. To have an accurate concept of what is "big", we have to know the market share of the devices to identify the distribution of the screen sizes.
– The concept "big" will change over time. What was considered a big screen in 2004 nowadays would be considered normal or small.
– It will also change with the location; a big screen in Europe probably is not so big in Japan.

```
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;
```

Fig. 5 Sample of fuzzy rules

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 Fig. 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 Fig. 5).

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). 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. To answer these questions, we must know the actual distribution of the market. Our proposed solution is to use popularity metrics to estimate the market share of the devices (in our case, we use Google Trends[7]). Besides, all the device models cannot have the same weight in the calculation,

[7] Google Trends. http://www.google.es/trends

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 1 year to another. 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 Europe. To tackle this problem, we support the geolocation of the results to filter them according to the needs of the developers.

The device characterization process can be divided en three different tasks: the initial data retrieval, the decay process, and the automatic membership function generation.

5.1 Initial data retrieval

The first step consists on retrieving and formatting all the necessary data. This means to parse the WURFL database and to retrieve the Google Trends data for all the devices. This is a lengthy process (it takes days to gather all the data) due to the IP address and account restrictions of Google Trends and must be done in a distributed way. The final result is a database with all the records of the trends for each taking into account the geolocation.

5.2 The decay process

Once we have all the data, the next step is to process the trend values to take into account their "age". Older trend values have less weight in the accumulated trend value of each device, reflecting the transitory nature of the mobile device market. This is done taking into account the phone plans of the principal telecommunication companies to try to model the mobile phone change cycle among users. We acknowledge that every user does not change its mobile phone at the end of the acquired mobile plan. We use the following values to calculate the decay:

- Last 15 months: take into account the 100% of the trend value.
- From 16 to 24 months: take into account the 90% of the trend value.
- From 25 to 36 months: take into account the 40% of the trend value.
- From 37 to 60 months: take into account the 10% of the trend value.
- More than 60 months: take into account the 5% of the trend value.

We would like to use a more robust model to calculate the decay, but as we would discuss in Sect. 6 we have found that this is a good approximation.

5.3 Automatic membership generation

Once we have the processed data (see Fig. 6 for an example of the distribution of the popularity of the different screen resolutions expressed as the multiplication of height and width) and the desired linguistic terms, we can automatically generate the membership functions for those terms.

The first step is to divide the data in regions (see Fig. 7) that will mark the point where each membership function will have its highest value. While creating the regions, the algorithm seeks to equally distribute the total trend value contained in each membership function, but usually, this goal is not achieved in the first iteration.

To solve this problem, we generate every possible permutation by moving each of the initial region boundary one step to each side. For each universe, we calculate its deviation from the ideal universe (see Fig. 8). First, we discard inconsistent universes following these rules:

- If the first region in the universe does not start in the 0 point, then that universe is discarded.
- If the left boundary of a region in the universe starts after the right boundary, then that universe is discarded.
- If the left boundary of a region starts before the right boundary of a previous region, then that universe is discarded.

Then, we calculate the deviation of each remaining universe. What we seek is to minimize the deviation from

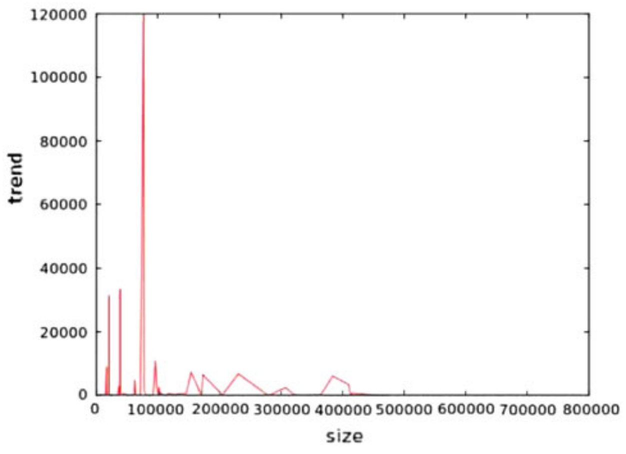


Fig. 6 Worldwide popularity calculation for all the resolutions of the mobile devices in WURFL

Fig. 7 Calculation of the base universe

```
function createBaseUniverse
Inputs:
  TRENDS is an association of key-value pairs where the keys are
  each distinct value of the target device feature, and the
  associated value is the summation of the trends for the
  devices with that feature value. The keys are ordered
  incrementally.
  NUM_OF_TERMS is the number of desired linguistic terms
Outputs:
    REGION_LIMITS is the location of the regions in the base
    universe

TOTAL_TRENDS ← $\sum_{i=0}^{n} VALUES(TREND_i)$

NUM_OF_REGIONS ← NUM_OF_TERMS - 1
IDEAL_TREND ← TOTAL_TRENDS / NUM_OF_REGIONS
ACCUMULATED_TREND ← 0
REGION_LIMITS ← empty set
CURRENT_VALUE ← first feature value in TRENDS

for CURRENT_REGION = 1 to NUM_OF_REGIONS do
    while ACCUMULATED_TREND <= CURRENT_REGION * IDEAL_TREND do
      CURRENT_TREND ← associated trend in TRENDS  to CURRENT_VALUE
      ACCUMULATED_TREND += CURRENT_TREND
      CURRENT_VALUE ← next feature value in TRENDS
    end while
```

Fig. 8 Calculation of the best possible universe

```
function findBestUniverse
Inputs:
  REGION_LIMITS is the location of the regions in the base universe
Outputs:
  BEST_UNIVERSE is the universe with the best fitness

ALL_UNIVERSES ← every possible permutation generated by moving one
                step the limits contained in REGION_LIMITS

for each PERMUTATION in ALL_UNIVERSES do
    if first region of PERMUTATION does not start in 0 then
        remove PERMUTATION from ALL_UNIVERSES
    for each REGION in PERMUTATION do
        if left limit of REGION > right limit of REGION then
            remove PERMUTATION from ALL_UNIVERSES
        elif left limit of REGION < right limit of previous REGION then
            remove PERMUTATION from ALL_UNIVERSES
    end for each
end for each

UNIVERSE_FITNESSES ← empty set

for each PERMUTATION in ALL_UNIVERSES
    PERMUTATION_FITNESS ← $\sum_{region \in PERMUTATION} |IDEAL\_TREND - \sum_{trend \in region} trend|$
    add PERMUTATION_FITNESS, PERMUTATION to UNIVERSE_FITNESSES
end for each
BEST_UNIVERSE ← first PERMUTATION in UNIVERSE_FITNESSES with lowest
PERMUTATION_FITNESS

return BEST_UNIVERSE
```

the ideal universe; thus, we select the universe with the lowest deviation.

Once we have found the best universe, we can finally build the membership functions for each linguistic term (see Fig. 9). To do this, we take into account that:

– The region boundaries mark the inflexion point from the ascending and descending curves of a linguistic term.
– The first linguistic term will only have a descending curve that will start in the left boundary of the first

Fig. 9 Calculation of the membership functions

```
function calculateMembershipFunctions
Inputs:
  BEST_UNIVERSE is the universe with the best fitness
Outputs:
  MEMBERSHIP_FUCTIONS the membership functions for the linguistic
  terms

LAST_CURVE ← empty set
MEMBERSHIP_FUNCTIONS ← empty set
for each REGION in BEST_UNIVERSE do
    ASCENDING_CURVE ← ac(x) = $\sum_{i=left\_limit}^{x} TREND_i \mid x \in feature\ values\ in\ REGION$
    DESCENDING_CURVE ← dc(x) = 1 - ac(x)  $\mid x \in feature\ values\ in\ REGION$
    CURRENT_CURVE ← LAST_CURVE + DESCENDING_CURVE
    add CURRENT_CURVE to MEMBERSHIP_FUNCTIONS
    LAST_CURVE ← ASCENDING_CURVE
end for

add LAST_CURVE to MEMBERSHIP_FUNCTIONS

return MEMBERSHIP FUNCTIONS
```

region and will end in the right boundary of the first region.

– The last linguistic term will only have an ascending curve that will start in the first boundary of the last region and will end in the right boundary of the last region.
– The ascending curves are calculated accumulating the trend values in a region.
– The descending curves are symmetrical to the ascending curves: dc(x) = 1 − ac(x), where ac is the ascending curve and dc the descending curve.

The result of this process can be seen in Fig. 10. Using the data shown in Fig. 6, we have divided it into 3 linguistic terms: small resolutions (dashed line), normal resolutions (solid line), and big resolutions (dotted line).

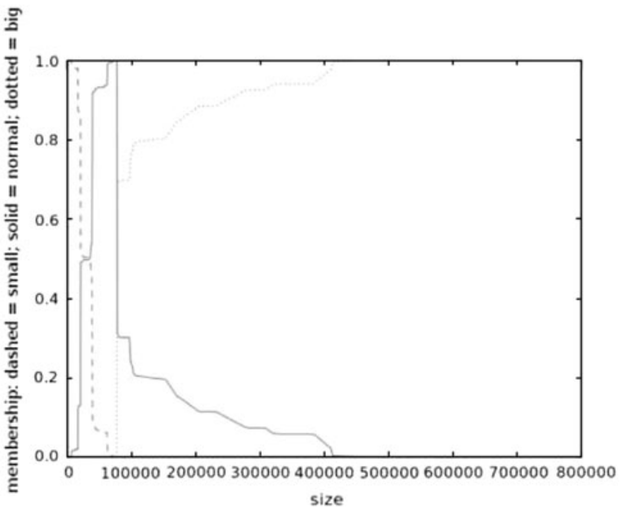


Fig. 10 Generated membership functions for the screen resolution

6 Evaluation

During the research of the project, two evaluations have been performed, aiming at different subareas. The first evaluation was performed to check the accuracy of the source used for establishing the relevance of the mobile devices to define the membership functions. The second evaluation was performed to measure the usability of the

Table 2 Membership values for different devices using worldwide data

Device	Resolution	Membership value for each term
Nokia 7110	96 × 65 pixels (4224)	Small: 0.91073068
		Normal: 0.08926932
		Big: 0
Nokia 6630	176 × 208 pixels (36608)	Small: 0.72814896
		Normal: 0.27185104
		Big: 0
Nokia N95	240 × 320 pixels (76800)	Small: 0
		Normal: 1
		Big: 0
HTC Hero	320 × 480 pixels (153600)	Small: 0
		Normal: 0.19230388
		Big: 0.80769612
Apple iPhone 3G	320 × 480 pixels (153600)	Small: 0
		Normal: 0.19230388
		Big: 0.80769612
Nokia N97	640 × 360 pixels (230400)	Small: 0
		Normal: 0.09409409
		Big: 0.90590591
Apple iPad	1,024 × 768 pixels (786432)	Small: 0
		Normal: 0
		Big: 1

overall system by performing a survey with software developers that used Imhotep in a small application.

6.1 Evaluating the source used for establishing mobile devices relevance

Due to the lack of hard data regarding the market share of each device, it is difficult to evaluate the results, but we can assess the goodness of fit of the created model comparing different results. In the first test, we compare the resolution size of different devices with worldwide popularity data taken until April 2010. We have divided the resolution into 3 linguistic terms: small, normal, and big.

As can be seen on Table 2 and Fig. 6, the most common resolution worldwide on March 2010 according to our system was 240 × 320 pixels. We have repeated the tests, but this time, we only used the Nokia N95 and N97 as target devices, and we have changed the location of the trends. As can be seen in Fig. 11, the generated membership functions change from one location to another. While the results for the worldwide and Spanish markets are similar, the membership functions generated for the Japanese market show some changes. The main difference is that there is a significant number of resolutions that can be considered mainly "big" worldwide and in Spain that are considered mainly "normal" in Japan.

This is more easily seen in Table 3. Comparing the results of the membership values for the Nokia N95 and the Nokia N97 in the three different markets, we can see that resolutions in Japan tend to be bigger than worldwide and Spanish mobile device's resolutions.

6.2 Evaluating the usability of Imhotep

In order to evaluate the usability of Imhotep, a survey was performed among software developers not involved in the project with the following questions:

1. What is your previous experience in the development of accessible or device adaptable applications? Being 1 "No experience" and 5 "Experienced".
2. How long did it take to learn how to use Imhotep? Being 1 "Much less than expected" and 5 "Much longer than expected".

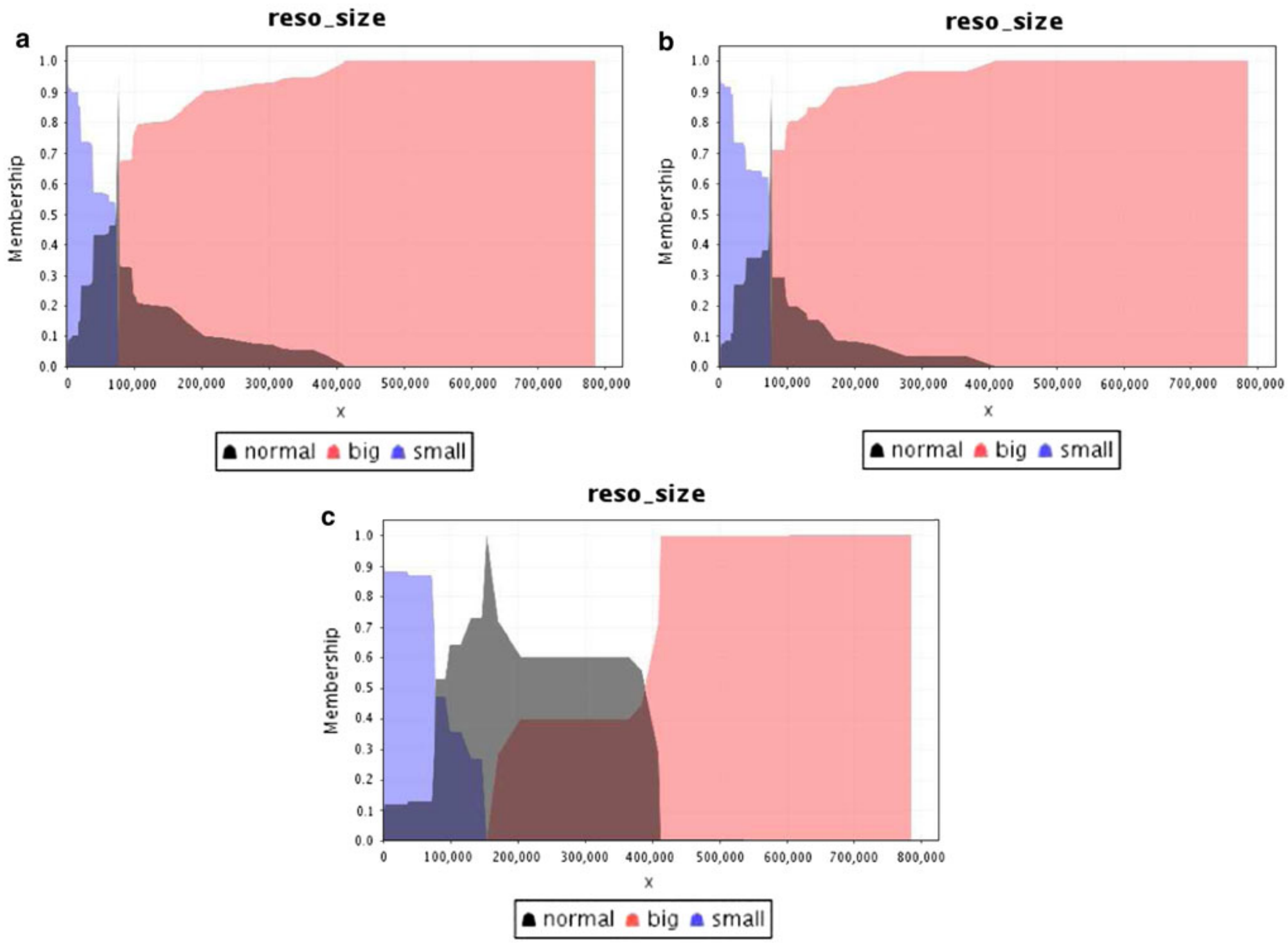


Fig. 11 Membership functions for the resolution size in different locations: **a** Worldwide, **b** Spain, **c** Japan

Table 3 Membership values for different markets

Device	Location	Membership value for each term
Nokia N95	Worldwide	Small: 0
		Normal: 1
		Big: 0
Nokia N95	Spain	Small: 0
		Normal: 1
		Big: 0
Nokia N95	Japan	Small: 0.66996612
		Normal: 0.33003388
		Big: 0
Nokia N97	Worldwide	Small: 0
		Normal: 0.09409409
		Big: 0.90590591
Nokia N97	Spain	Small: 0
		Normal: 0.07021852
		Big: 0.92978148
Nokia N97	Japan	Small: 0
		Normal: 0.60196775
		Big: 0.39803225

3. How complex you found using preprocessor directives? Being 1 “Very simple” and 5 “Very complex”.
4. How much does Imhotep facilitate the development of accessible or device adaptable applications? Being 1 “It makes it more complex” and 5 “It facilitates very much the development”.
5. How useful did you find Imhotep? Being 1 “Useless” and 5 “Very useful”.
6. How long did it take to perform the test? Being 1 “Much less than expected” and 5 “Much longer than expected”.
7. Would you use Imhotep for future accessible or device adaptable applications? Being 1 “Very unlikely” and 5 “Very likely”.

The results are presented in percentage in Table 4, highlighting those with highest votes for each question. As it can be appreciated, 75% of the developers considered that Imhotep facilitates much or very much the development of accessible applications, and 88% considered that it was useful or very useful. The time required for learning to use was considered very low, and developers considered the directives simple enough, although the time required to perform the test was considered medium. Finally, most developers considered Imhotep interesting for future accessible or device adaptable applications developments.

Table 4 Results in percent of the usability evaluation

Question	1 (%)	2 (%)	3 (%)	4 (%)	5 (%)
1	**63**	13	25	0	0
2	**38**	13	**38**	13	0
3	**75**	0	13	0	13
4	13	0	13	**50**	25
5	0	0	13	**63**	25
6	25	13	**38**	25	0
7	0	13	0	**75**	13

7 Use case

To prove the capabilities of the Imhotep framework, we have developed an adaptable application aimed to two different user groups: people without disabilities and blind people. The developed application helps people finding useful places (monuments, restaurants, banks...) around them using augmented reality. The user interface requirements of both groups are completely different. In the first case, the augmented reality will work in the traditional way, superimposing the directions to the place in the images taken by the camera. But for the blind users, this is not a valid alternative. In this case, the interface will take a multimodal approach to the accessibility. First, it will provide a voice interface that will guide the blind person through the menus and while he moves the mobile device to find the direction of the selected place. Second, it will also provide a text only interface in case that the user prefers to use its own screen reader. Finally, it will provide feedback using the vibration capabilities of the mobile device. The different interfaces can be seen in Fig. 12. The left column shows the interface for blind people, with the text-only mode for screen readers. The right column shows the UI for people without disabilities, featuring an augmented reality interface.

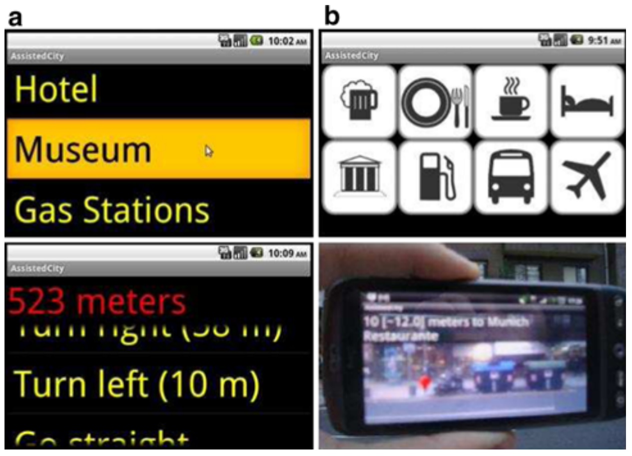
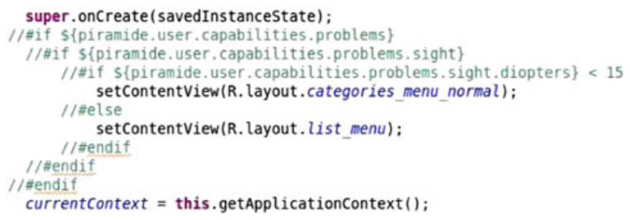


Fig. 12 Different user interfaces for: a Blind people and b People without disabilities



```
  super.onCreate(savedInstanceState);
//#if ${piramide.user.capabilities.problems}
  //#if ${piramide.user.capabilities.problems.sight}
      //#if ${piramide.user.capabilities.problems.sight.diopters} < 15
          setContentView(R.layout.categories_menu_normal);
      //#else
          setContentView(R.layout.list_menu);
      //#endif
  //#endif
//#endif
  currentContext = this.getApplicationContext();
```

Fig. 13 Snippet of the preprocessor directives used in the augmented reality application developed using the Imhotep Framework

Using our framework, the code (a small fragment of the code can be seen in Fig. 13) only has to be written once. The code will be adapted to the target user and platform in the preprocessing step, deleting the unnecessary parts.

8 Conclusions and further work

The adoption of technology among elders and people with disabilities has always been a slow process. In many cases, this is due to the difficulties encountered by these groups when using them. Technology in general and mobile devices in particular offer great opportunities for these user groups, helping them to improve their quality of life and increasing their independence. This is why is important that those applications aimed to these collectives are easy to use and adapted to their capabilities, but in many cases, we are too focused on the technology and forgot people.

Trying to resolve this situation, we have presented in this paper 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). We have evaluated our approach to adaptable interfaces by performing a test with a group of mobile application developers, showing that Imhotep framework facilitates the creation of accessible applications.

Future work will focus on making Imhotep more developer friendly with the integration of the framework with an IDE. We will also like to improve the membership function calculation process, trying to create a more robust decay model. Finally, we plan to test our framework with a group of mobile application developers to evaluate its capabilities and identify possible improvements.

Acknowledgments This work has been supported by project grant TSI-020301-2008-2 (PIRAmIDE), funded by the Spanish Ministry of Industry, Tourism and Trade.

References

1. Kaye HS (2000) Computer and internet use among people with disabilities. In: Disability statistics report 2000. Department of Education, National Institute of Disability and Rehabilitation Research, Washington, DC
2. Walker A (1999) Ageing in Europe—challenges and consequences. Zeitschrift für Gerontologie und Geriatrie (Springer Berlin/Heidelberg) 32(6)
3. Ambient Assisted Living Joint Programme (2010) http://www.aal-europe.eu
4. Ahn H-I, Lee J-E, Yoon Y-I (2007) A middleware for user centric adaptation based on fuzzy in ubiquitous environment. Sixth international conference on advanced language processing and web information technology
5. Abe M, Morinishi Y, Maeda A, Aoki M, Inagaki H (2009) Coool: a life log collector integrated with a remote-controller for enabling user centric services. Digest of Technical Papers International Conference on Consumer Electronics
6. Al Masum SM , Prendinger HT, Ishizuka M (2007) Emotion sensitive news agent: an approach towards user centric emotion sensing from the news. IEEE/WIC/ACM international conference on web intelligence
7. Hagen S, Sandnes F (2010) Toward accessible self-service kiosks through intelligent user interfaces. Pers Ubiquitous Comput 14(1):1–7
8. Hervás R, Bravo J, Fontecha J (2010) A context model based on ontological languages: a proposal for information visualization. J Univers Comput Sci (JUCS) 16(12):1539–1555
9. de Ipiña DL, Vázquez JI, García D, Fernández J, García I, Sainz D, Almeida A (2006) EMI2lets: a reflective framework for enabling AmI. J Univers Comput Sci (JUCS) 12(3):297–314