Flippable User Interfaces for Internationalization

Iyad Khaddam and Jean Vanderdonckt
Université catholique de Louvain, Louvain School of Management
Louvain Interaction Laboratory, Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{iyad.khaddam, jean.vanderdonckt}@uclouvain.be – Phone: +32 10 478525

ABSTRACT
The language reading direction is probably one of the most determinant factors influencing the successful internationalization of graphical user interfaces, beyond their mere translation. Western languages are read from left to right and top to bottom, while Arabic languages and Hebrew are read from right to left and top to bottom, and Oriental languages are read from top to bottom. In order to address this challenge, we introduce flippable user interfaces that enable the end user to change the reading direction of a graphical user interface by flipping it into the desired reading direction by direct manipulation. This operation automatically and dynamically changes the user interface layout based on a generalized concept of reading direction and translates it according to the end user's preferences.

Author Keywords
Adaptation, cultural background, internationalization, reading direction, user interface layout.

General Terms
Design, Experimentation, Human Factors, Verification.

ACM Classification Keywords
D2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces*. D2.m [**Software Engineering**]: Miscellaneous – *Rapid Prototyping; reusable software*. H.5.1 [**Information interfaces and presentation**]: Multimedia Information Systems – *Animations*. H5.2 [**Information interfaces and presentation**]: User Interfaces – *Graphical User Interfaces (GUI); User-centered design, Windowing Systems*.

INTRODUCTION
Localization usually refers to the process of designing and developing a Graphical User Interface (GUI) that is adapted to a particular culture [6,13], continent [4], country [14] or region [18], or a set of them (based on [4]). The opposite process, called *globalization*, usually refers to the process of designing and developing a GUI that accommodates the common ground of the largest possible audience from different cultures, continents, countries, and regions (based on [4,18,21]. Making a UI *global* [21] therefore results into one single GUI, while making a UI *local* results into many different variations of the same initial GUI, but adapted to the different cultural backgrounds. These different variations are subject to a series of challenges: their design should be coordinated [4], culturally-aware [13], maintained simultaneously [18], and applicability of a change request depending on local settings. A single change request may indeed affect one or several variations of the same GUI. Many different factors could positively influence a successful localization [4,6,10,11,13,14,15,16,18,21]: color, format, metaphor, screen, layout, language, images, structure, density, ordering of information,….

Instead of producing several variations of the same initial GUI for the different localizations, it might be interesting to concentrate the adaptation logic into a single GUI that handles these variations depending on setting of the end user. A single GUI could be produced with adaptation, thus addressing the challenge of coordination and simultaneous maintenance, but leaving the challenges of culture awareness and dependability open in the adaptation logic.

One of the main critical factors of success is the adaptation of the GUI to the end user's language, which includes translation (e.g., by automated translation of all contents and resources [15]) and layout adaptation to the language reading direction [8]: this adaptation largely fosters the UI acceptance [8], other aspects, such as color, font, and size, are mostly lexical factors, and less critical, while high-level aspects, such as metaphors and organization are hard to predict in a systematic way [18] and their impact depends on many cultural parameters that are hard to reproduce [14].

This paper introduces the interaction technique of *flippable user interfaces* in order to support the adaptation to the end users' language, which subsumes translation, transformation of the layout to support a correct reading direction, mixing different directions, improving the visual properties of the layout (such as balance and symmetry [22]).

The remainder of this paper is structured as follows: the next section reports on some related work. Then, the design principles that underlie flippable user interfaces (i.e., based on a concept of generalized direction) are introduced, motivated, and exemplified. The software architecture supporting the implementation of flippable user interfaces as an adaptation interaction technique for addressing internationalization is discussed. Finally, a conclusion delivers the main points of this research and presents some future avenues, especially for the end user's acceptation.

EICS'11, June 13–16, 2011, Pisa, Italy.

RELATED WORK

This paper is aimed at developing an interaction technique (i.e., flippable user interface) as a support for internationalization (i.e., adaptation to end user's language) of GUIs with transition (i.e., animated transition between UI before and after adaptation). The following state of the art is structured with respect to these three main fields of research.

Adaptation to Cultural background

Three kinds of GUI adaptation are usually performed in order to localize a GUI: *technical adaptations* [18] that address the needs for making the GUI workable and displayable in the localized context of use (e.g., by use of appropriate alphabet, character set), *national adaptations* that address the needs of particularizing the information and their associated actions to a particular country (e.g., by adding information relevant to a country only, by removing unnecessary menu items for a particular task that does not require it in a specific country), and *cultural adaptations* that address the needs of cultural habitudes, conventions, and meanings [13,14]. While most of the adaptation operations are well documented in the literature, they are applied mostly on a case by case way. They are rarely applied systematically or encapsulated in an adaptation engine.

The main goals for adaptation towards localization are [8, 14,18]: communicate in the country's native language; support the natural writing symbols, punctuation, and so on; support native date, currency, weight scales, numbers and addresses; support natural work habits and the work environment, and communicate in an inoffensive manner. Again, while these principles are largely recognized and widely adopted, they are seldom translated into rules that automatically transform GUIs for a localized purpose.

In the market, we can find well-Arabized products and most of them are built on Microsoft Windows Operating System, as it was the pioneer in providing support for right-to-left (RTL) languages or on the web. ERP products were forced to provide RTL support due to market pressure.

Portenari & Amara explain the Arabic language characteristics and explain the challenges of the language in the context of providing OS support for Arabic. They discuss the issues related to encoding, character shaping and the "cursive" or "handwritten" style of writing in Arabic, vowels, numbers shapes and the mirroring effect on visual screens.

Rejmer *et al.* [18] discussed the internationalization of a web site that performs automated evaluation of W3C accessibility guidelines based on a set of design guidelines for supporting internationalization/localization. These guidelines only address Western languages (Latin alphabet), thus ignoring other reading directions, but identifying important GUI properties that are affected by internationalization.

XUL supports RTL UI [16] by providing the "dir" property at the UI element which is the base of all elements. The "dir" property can have one of two values: normal, reverse. The "normal" means: "position elements in the container according to their order in the xml file". The "reverse" value means: "position elements in the container according to their *reverse* order in the xml file". XUL does not directly address the RTL concept, but their concept of "reverse" fixes the orientation issue. On the other side, it does not address the control localization nor provide support to it. When the "dir" property is "reverse", this doesn't imply that the final control to be used is a control that supports RTL. XUL depends on the rendering framework to determine the final control (localized version). Therefore, it provides a localized version of Firefox for each language. The problem we note with XUL approach is that each localized version of the product will have a localized version of the design (the XUL file is copied for each language). This imposes a maintenance/update problem.

Quiroz *et al*. [16] implemented a genetic algorithm for automated generation of GUI layouts based on user fatigue based on the XUL language. By applying this algorithm, they demonstrate which layout causes the less fatigue, but again, the layout if only LTR since XUL itself is like that.XAML (www.xaml.net/) is a markup system that underlies user interface components of Microsoft's .NET Framework 3.0 and above. XAML supports RTL by adding a "FlowDirection" property to the containers and UI elements that takes one of the values: "LeftToRight", "RightToLeft". This causes the expected effect of RTL to be applied on the container and/or the element. Figure 4 gives an example.

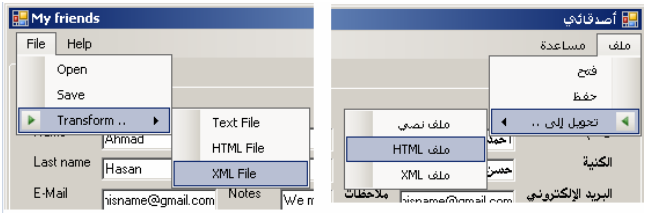

Figure 1. A sample LTR and the localized RTL version (in Arabic language).

Interaction techniques

Various interaction techniques have been investigated in Human-Computer Interaction (HCI) that are related to the *metaphor of flipping a page of a book*. This metaphor has been extensively used in hypermedia and hypertext applications since a long time (e.g., in HyperCard) and is still used today in multimedia presentations for the following reasons: (i) the flipping gesture is familiar with the activity of browsing a book, an album (e.g., *FlipAlbum* [7]), or a stack of pages or documents since a simple flip distinguishes forward from backward movement, (ii) the flipping gesture is natural and straightforward to produce, (iii) flip, drop, and turn, are basic operations of geometric symmetry [22] that respectively reflect an image around the *y*-axis, the *x*-axis, perform a 90° rotation to the right, and (iv) flipping a window could reveal additional information related to the window (e.g., as in Sun's Looking Glass 3D desktop). In other words, the flipping gesture indicates a direction, which is appropriate to denote the direction of reading. The

flipping metaphor has however different interpretation: for Western countries where the page is flipped from right to left in order to support the reading direction from left to right and top to bottom. '*Fold and Drop*' [5] is an interaction technique enabling end users to drag an icon from a stack of overlapping windows and drop it onto a possibly hidden window by applying gestures on the windows, thus releasing the user to constantly switch from one window to another. *Orimado* [9] is a variant of this interaction technique for Oriental languages. While Fold & Drop and Orimado also rely on the metaphor of flipping, they are applied to window and maintain the direction from right to left without affecting the UI contents since this is not their goal.

The idea of rotated and peeling the windows (Figure 2), and snapped and zipped windows has already being introduced as an interaction technique recommended for manipulating multiple windows more efficiently [1]. Preliminary investigations [1,23] show that this interaction technique generates a high subjective satisfaction rate, not just because it is graphical or easy to use, but mainly for its metaphor that is pretty close to the real world.

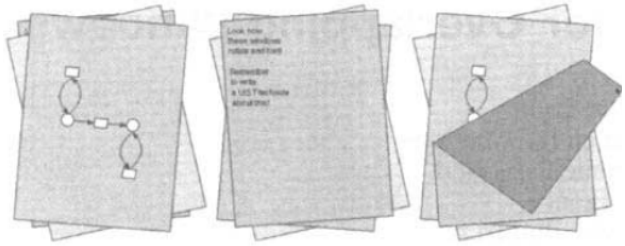

Figure 2. Example of the rotated and peeled-back metaphors.

Flip zooming [3] is an interaction technique that consists of splitting a screen into a sequence of objects (e.g., images, fragments of texts, or combinations of both types), putting the focus on of these objects, and letting the end user to flip through the sequence of objects from left to right, top to bottom to preserve the context. Zooming in/out is then used on any object of interest. The main drawback of this technique is a constant 'touch-and-go' between flipping (to navigate) and zooming in/out (to see the details).

Flying [12] is an interaction technique used for quickly browsing a large hypertext in order to gain some insight to features such as organization, size, depth, level of detail, and layout based on flipping. This technique is not intended to support reading the contents, but to provide a first idea of how it is structured, even in a non-linear order.

As we can see from existing work, it seems interesting to consider the concept of flippable UI that mimics paper-based operations in order to improve the end user's subjective satisfaction and naturalness of interaction. While different operations that mimic page flipping have been introduced, none of them really reproduce the flipping of a page in any direction. that results into an adaptation to the new state. This is why animated transitions are made for: convey the change of display.

Animated Transitions

It is well established [2,7,21,23] that it is generally considered easier for users to maintain a mental model of the data across smooth transitions and less time is spent comprehending the new data presentation. In other words, an animated transition between two states is appreciated because it supports a progressive transition from the initial state to the final state without any disruption between. Several visual techniques exist that could be applied to GUI design, whether it is for one localization [23] or for globalization of these GUIs [4,22], the two main dimensions of internationalization. Usability guidelines (e.g., [4,13,14,18]) also exist that reply on animated transition to change the state of a GUI depending on its culture [11].

DESIGN PRINCIPLES OF A FLIPPABLE USER INTERFACE FOR INTERNATIONALIZATION

In this section, we motivate, define, and discuss the design principles that led us to rely on flipping, an interaction technique augmented by an animated transition, as a way to support the change of cultural background (here, mainly Western vs. Eastern cultures, with various ways of reading).

Principle #1. Provide handles for direct flipping

In order to adhere to the principles of the direct manipulation, it is expected that the flipping interaction technique should be supported by handles (Figure 3) that indicate the direction where to flip: left to right (LTR), right to left (RTL), top to bottom (TTB) or bottom to top (BTT). Each flipping direction then indicates the reading order of the GUI, thus provoking its adaptation to the corresponding cultural background. In direct manipulation the objects should have a graphical representation, preferably one that is close to the real world if any, with an incremental interaction that is fast and reversible. For this purpose, horizontal handles (represented in red in Figure 3) control transformations along the *X* axis, while the vertical handles (represented in blue in Figure 3) control transformations along the *Y* axis. An arrow-shaped handle may convey a translation, a square-shaped handle may convey a change of scale, and a circle-shaped handle (represented in green in Figure 3) may convey a rotation. Similarly for a 3D object in space, these three handles could support respectively *nutation* (along *X* axis), *precession* (along the Y axis), and *rotation* (along the Z axis). For instance, The Card Stack had markers on front side and back side, so the user was able to flip it backward, and saw the backside of the 3D model. The central button, represented in grey in Figure 3, restores the transformation to its initial stage or identity. The ellipse and the lines depict the current status of transformation.

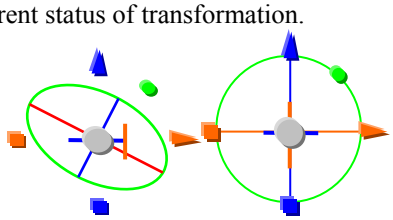

Figure 3. Handles for direct flipping.

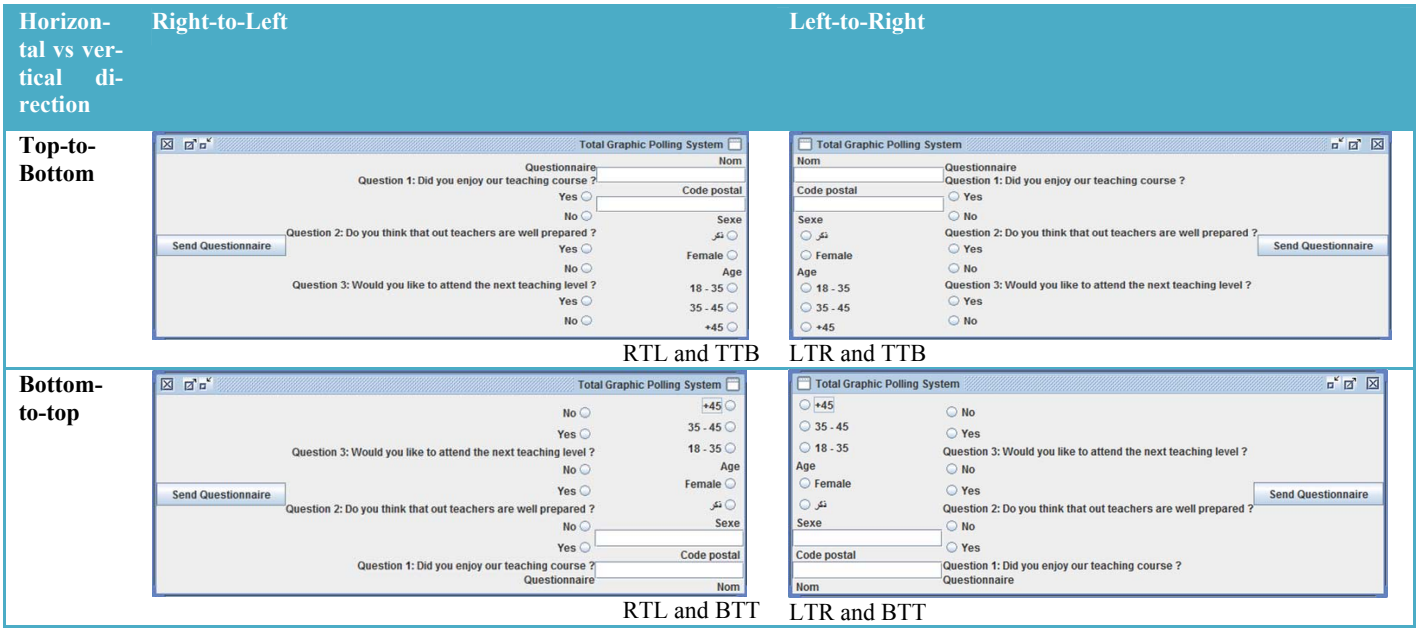

Figure 4. The four possible layouts depending on the generalized direction.

Principle #2. Perform adaptation according to flipping

From the aforementioned literature, we summarize the UI adaptation in basically two ways:

- The *orientation* (the mirroring): this consists of changing the reading direction of the contents depending on the end user's native language. Some languages are read LTR while some others are read RTL. In order to generalize this attribute, we also introduce the vertical dimension since some languages are read TTB or BTT. Therefore, each native language is assigned to a reading direction according to the horizontal dimension (LTR vs TRL) and the vertical dimension (TTB vs BTT), thus covering the four possible cases (Figure 4). The common direction in Western countries, e.g., LTR and TTB, is reproduced in the top right position of Figure 4. If the GUI is flipped to the left, it reverses the order from LTR to RTL; if the GUI is flipped to the bottom, it reverses the order from TTB to BTT. Figure 4 keeps the same contents and language in order to remain understandable. Automatic translation of the language [15] is a service that could be used for translating the labels, instructions, messages from one language to another, even at run-time. After changing the layout, several operations are required by the language change.
- The *localization*. The TRL localization has special characteristics that do not exist in Latin-alphabet languages, which makes the localization a challenge. We identify from the examples before the following characteristics for RTL localization:
  - *Text* localization:
    - Text localization: language encoding and character set (alphabet).
    - Direction switching: direction of text writing.
  - *Graphics* localization:
    - RTL sensitive graphics (non-horizontally symmetrical)
    - Images with text inside
    - Other localizable images (country flag…)
  - *Control* localization:
    - Control rendering: for instance, a label control should support writing from right to left.
    - Control behaviour: controls should be aware of special behaviour for special keys, like pressing "enter" key in a text area.

Principle #3. Ensure smooth transition for adaptation

To avoid startling and confusing users, we employ smooth slow-in and slow-out transitions [21,23] for every visual change occurred after the adaptation has been performed. This animated transition has the advantage to preserve most of the visual aspects of GUI widgets horizontally or vertically. Other animated transitions could be investigated depending on the adaptation operation that has been executed, thus opening a door for many different ways to ensure a smooth transition between the GUI before and after adaptation to a cultural background (here, localization). Animated transitions however suffer from the "lag" problem [21] that may cause end user frustration if the animated transition is too fast or too slow. Here, a flippable user interface does not suffer too much from the "lag" problem because the animated transition is performed at run-time while the end user is flipping the GUI. In the next section, we describe how this engineering technique has been implemented. Flipping at run-time is the target because a user may speak different languages and may want to switch from one language to another, because the "by default" language is not always right, even in an option menu, because a UI may contain various data simultaneously in different formats and languages and because manually developing different layouts for different languages requires extensive development efforts that are not required by our approach.

IMPLEMENTATION

This section describes how the three aforementioned design principles have been implemented together in order to make the concept of flippable user interface operational. The software architecture is structured as follows (Figure 5):

1. **The Widget Selector**: This component is responsible of mapping the nodes of a GUI specified in a User Interface Description Language (UIDL) to output components. It needs the mapping XML file. Each specific output type (swing, html) needs a specific map.xml, but the Widget Selector is the same for all maps. In our case, UsiXML (www.usixml.org) was used for testing, but other similar UIDLs could be used in the same way. A XML map is structured as follows: the root is usiMapModel. This contains sub elements "<cuiMap>" that determine how we will map UsiXML tags in the CuiModel section of the xml file. Element selection algorithms are simply saying: a set of conditions that determine the element selection. The control structure in the usiMapModel is the "<option>" tag. This tag has 2 sections: condition and action. The condition is a simple Boolean expression that can be written in groovy (a java like syntax, www.groovy.com) which is modeled with a "<condition>" tag in usiMapModel file. The actions are a set of commands that create the suitable widget and fine tune its properties. They are modeled using the "<maps>" tag. This tag contains an init attribute, and that is where we usually create the appropriate widget. The child node "<map>" is a statement to map a property from a Usi node to the mapped widget property.
2. **The CUI Tree Traversal:** This is simply a tree traversal algorithm. Currently, we implemented a Depth-First traversal algorithm for parsing the Concrete User Interface (CUI). If some type of rendering needs to consider Width-First traversal, then the Width-First class needs to be implemented and passed to the engine instead of the default traversal (Depth-First).
3. **The Merge Algorithm:** this is simply a component that merges the tree of components in a custom way suitable to the final output. In the case of Swing, the components are all of type *java.awt.Component*, and the merge algorithm will simply add Components to the Container (a sub class of Component) but calling the method add.
4. **The CuiRendering Engine:** The engine that orchestrates the messages among the above components. The engine calls the traversal to start traversing the UiModel, which will parse the UiModel tree looking for UsiXml nodes. For each node, it will notify the engine who will call the Widget Selector to resolve this node. If the map.xml file contains a mapping for this node, it will create the mapped component and return it to the engine, who in turn will store it a special storage called: UsiRuntime. The UsiRuntime contains a tree of rendered objects. The UsiRuntime allows us to retrieve the rendered component by using the "id" of the relative UsiXml element. This allows 2-ways mapping between original UsiXml element and the rendered element. When finished, the engine will call the Merge Algorithm to merge the resulting tree of components and returns the UsiRuntime object to the caller. The UsiRuntime now contains the roots components (if the UsiXml file contains 2 Window elements, UsiRuntime will contains 2 roots, and so on). The caller uses these roots for any purpose, e.g., display on screen, save.

Figure 6 shows how the control panel could be displayed for finely governing the layout of the GUI. By moving the handles according to the Principle #1, the end user is able to manipulate the geometry of the GUI in order to reverse its reading order, but also in order to accommodate constraints imposed by the screen resolution. This engineering technique therefore supports the concept of variable geometry layouts that could accommodate different physical constraints imposed by the target computing platform. A *user interface with variable geometry* is hereby defined as a GUI exhibiting the capability of altering its layout geometry by direct manipulation depending on the context of use, thus making it *plastic* to some extent.

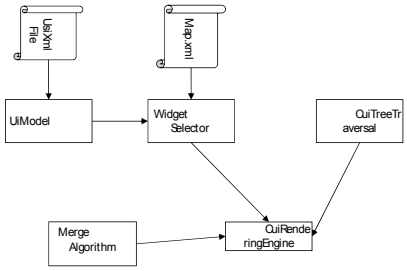

Figure 5. The software architecture of the flippable UI.

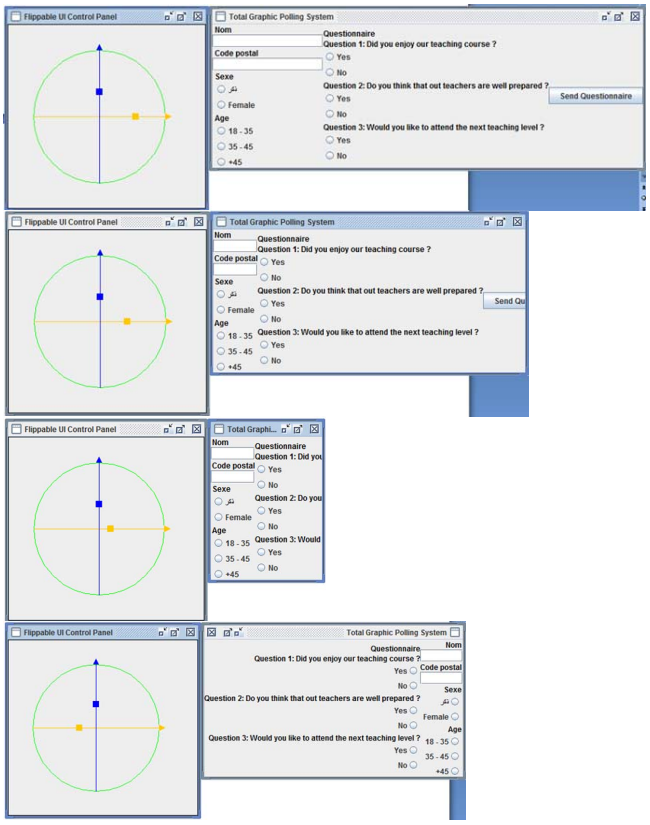

Figure 6. The control panel of the flippable UI.

When the end user does not want to finely manipulate the layout geometry, she could only flip the GUI in the desired direction, thus obtaining only the four possibilities depicted in Fig. 4, along with adaptation techniques executed as in Principle #2. Principle #3 preserves the continuity between the GUI before flipping and after flipping.

CONCLUSION

This paper presented the concept of flippable user interface as an interaction technique for supporting internationalization of graphical user interfaces. Traditionally, internationalization of a GUI is achieved by providing different sets of translated resources that are then incorporated in the executable code of the interactive application that is then responsible for switching to one layout to another. This method is largely *static* (all resources should be provided in resource files, translated and compiled) and *physically defined* (all layouts are done manually and embedded in the code). Instead, a flippable user interface introduces an engineering technique where the GUI definition is dynamically parsed, interpreted, and computed depending the default language and country settings of the end user. Switching from one cultural background to another, e.g., switching from English to Arabic is done performed dynamically by flipping the window by the end user in the desired reading order at runtime. Such a flippable UI could be flipped in any direction, vertical or horizontal. All other operations that are subsequent to a change of cultural background are then dynamically performed for all widgets, like localization of controls, text, messages, labels, etc. This approach is much more flexible both for the end user since there is no need to select parameters from a menu and for a designer/developer since all the variations are made flexible in the interpretation mechanism.

ACKNOWLEDGMENTS

The authors would like to acknowledge the support of the ITEA2-Call3-2008026 USIXML (User Interface extensible Markup Language) European project and its support by Région Wallonne DGO6 as well as the FP7-ICT5-258030 SERENOA project supported by the European Commission.

REFERENCES

1. Beaudouin-Lafon, M. Novel Interaction Techniques for Overlapping Windows. In *Proc. of UIST'2001*. ACM Press, New York (2001), pp. 153–154.
2. Bederson, B.B. and Boltman, A. Does Animation Help Users Build Mental Maps of Spatial Information? In *Proc. of IEEE Symposium on Information Visualization InfoVis'99*. IEEE Computer Society Press, Los Alamitos (1999), pp. 28–35.
3. Björk, S., Holmquist, L.E., Redström, J., Bretan, I., Danielsson, R., Karlgren, J., and Franzén, K. WEST: A Web Browser for Small Terminals. In *Proc. of UIST'99* (Asheville, Nov. 7-10, 1999). ACM Press (1999), pp. 187–196.
4. DelGaldo, E.M. and Nielsen, J. *International User Interfaces*. John Wiley & Sons, New York (1996).
5. Dragicevic, P. Combining Crossing-Based and Paper-Based Interaction Paradigms for Dragging and Dropping Between Overlapping Windows. In *Proc. of UIST'2004* (Santa Fe, October 24-27, 2004). ACM Press, New York, pp. 193–196.
6. Evers, V. and Day, D. The role of culture in interface acceptance. In *Proc. of INTERACT'97* (Sydney, July 14-18, 1997). Chapman & Hall, London (1997), pp. 260–267.
7. FlipAlbum, http://www.flipalbum.com.
8. Halawani, S.M. The effect of language reading direction on user interface design. PhD thesis, 1996.
9. Inaba, K., Orimado (2010). http://www.kmonos.net/lib/orimado.en.html. Visited on: December 5th, 2010.
10. Ishida, R. Creating HTML Pages in Arabic, Hebrew and Other Right-to-left Scripts. W3 Consortium, Geneva (2002). http://www.w3.org/International/tutorials/bidi-xhtml/. Visited on December 5th, 2010.
11. Khaddam, I. and Vanderdonckt, J. Adapting UsiXML User Interfaces to Cultural Background. In *Proc. of 1st Int. Workshop on User Interface eXtensible Markup Language UsiXML'2010* (Berlin, 20 June 2010). Thales Research and Technology France, Paris (2010), pp. 163–170.
12. Lai, P. and Manber, U. Flying Through Hypertext. In *Proc. of 3rd ACM Conf. on Hypertext Hypertext'91* (San Antonio, Dec. 15-18, 1991). ACM Press, New York (1991), pp. 123–132.
13. Mahemoff, M. and Johnston, L. "Culturally-aware" requirements for internationalized software. In: *Proc. of 3rd Australian Conf. on Requirements Engineering ACRE'98* (Geelong, October 26-27, 1998). D. Fowler, L. Dawson (eds.), Deakin University, Geelong (1998), pp. 83–90.
14. Marcus, A. and West Gould, E. Cultural Dimensions and Global Web User-Interface Design. *Interactions* 7, 4 (July 2000), pp. 32–46.
15. Pérez-Quiñones, M.A., Padilla-Falto, O.I., and McDevitt, K. Automatic language translation for user interfaces. In: *Proc. of the Int. Conf. on Diversity in computing TAPIA'2005* (Albuquerque, October 19-22, 2005). ACM Press, pp. 60–63.
16. Portaneri, F. and Amara, F. Arabization of Graphical User Interfaces. In [2], pp. 127–150.
17. Quiroz, J.C., Louis, S.J., and Dascalu, S.M. Interactive evolution of XUL User Interfaces. In *Proc. of the 9th Annual Conf. on Genetic and evolutionary computation GECCO'2007*. ACM Press, New York (2007), pp. 2151-2158.
18. Rejmer, P., Cooper, M., and Vanderdonckt, J. Lessons Learned From Internationalizing a Web Site Accessibility Evaluator. In *Proc. of 3rd Int. Workshop on Internationalisation of Products and Systems IWIPS'2001 "Designing for global markets 3"* (Milton Keynes, July 11-14, 2001). Digital Printing Services, Milton Keynes (2001), pp. 61–79.
19. Sears, A. Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout, *IEEE Transactions on Software Engineering 19*, 7 (July 1993), 707 - 719
20. Schlienger, C., Conversy, S., Chatty, S., Anquetil, M., and Mertz, Ch. Improving Users' Comprehension of Changes with Animation and Sound: An Empirical Assessment. In *Proc. of Interact'2007*. LNCS, Vol. 4662, Springer, pp. 207–220.
21. Stasko, J. Animation in User Interfaces: Principles and Techniques. In *Proc. of User Interface Software '93*, pp. 81–101.
22. Taylor, D. Global software: Developing applications for the international market. Springer-Verlag, Berlin (1992).
23. Vanderdonckt, J. and Gillo, X. Visual Techniques for Traditional and Multimedia Layouts. In: *Proc. of 2nd ACM Workshop on Advanced Visual Interfaces AVI'94* (Bari, 1-4 June 1994), ACM Press, New York (1994), pp. 95–104.