IntNovate a Toolkit to Ease the Implementation of Every Interaction Paradigm on Every Device

Bruno Merlin

Universidade Federal do Pará, FACE, Cametá, Brazil
brunomerlin@ufpa.br

Abstract. With the evolution and diversification of devices and platforms, we observed an evolution of the interaction paradigm usage, but also the emergence of several specific SDKs and toolkits. We present a toolkit, IntNovate, aiming at facilitating every interaction techniques and every interface paradigms in a large set of devices. The toolkit enables to create traditional widget applications, but also incorporates gaming techniques to turn easy animation integration, see-through interactions and direct manipulations. It is compatible with J2SE, J2EE, J2ME and android environments. A first evaluation compared an HMI development using both J2SE and IntNovate none form based application development and illustrated the IntNovate advantages in this context.

Keywords: Toolkit, graphic, direct interaction, multiple devices, multiple platforms.

1 Introduction

During the last decade, the graphic design became a major factor to improve software usability so as to increase software commercial impact. Consequently, the HMI design of commercial software turned to a multidisciplinary task involving HMI developers, ergonomists and graphic designers. In the same time, the number and kind of new devices grew significantly thanks to the evolution of mobile devices and to the integration of computation skills into the traditional appliances (household devices, car dashboard, electronic cash, etc.). At last, the device diversification increased the user's number and amplified their heterogeneity.

The diversity of devices and the evolution of operating systems, the intervention of new actors into the design process, the requirement to turn the software graphically attractive, and the needs to lead with heterogeneous users, changed the way to design graphic interfaces. We observed two mains evolution axes: a convergence between mobile and desktop interaction paradigms and operating systems [14, 15]; and the use of gaming interaction techniques into standard applications. It increased the use of direct interaction [13, 4], animations, gestural interaction [5], marking menu [8], see-through interfaces [3], etc. Graphic interfaces tend to leave their standard rectangular layout and interactive components are frequently integrated into a scene.

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2014, LNCS 8510, pp. 331–339, 2014.

In spite of these convergences, the HCI development toolkits remain specific for desktop, web or mobile environment, and do not ease the incorporation of gaming techniques into standard applications.

In this article, we present a HCI toolkit, IntNovate (www.intnovate.org), responding to these problems. The Toolkit is developed in Java and offered an implementation for J2SE, J2EE, J2ME and Android enabling to attend the majority of web, desktop and mobile platforms. The toolkit proposes several predefined behaviors, animation motor and skinable widgets that enable to create a compact and high expressive code such as actionscript, whereas beneficing of the java inter-operability. IntNovate also aims at easing the cooperation between developers and graphic designers by loading the graphic scene from vector graphic files generated by drawing and design tools.

In the next chapters, we detail the characteristics of the toolkit, the design process using IntNovate and present a first evaluation of the toolkit aiming at demonstrating its usability.

2 The Toolkit

2.1 Graphical Properties

Such as the majority of drawing toolkits, IntNovate is based on a 2D canvas managing RGB colors, opacity, textures and clipping. The graphical objects composing the application are organized into a single tree. The leafs represented the rendered primitive graphical objects (such as path with one or several contours, ellipses, texts, etc.). The branch nodes manage default properties values for the sub-tree nodes (such as fill and stroke brush, etc.). A local transformation matrix, an opacity coefficient, and a clip may be applied to every node. The tree is rendered by a depth-first traversal from the interface root node. Local node transformation, clip and color opacity are composed with the transformation, clip and opacity of parent nodes.

In fact, displaying a graphical component of the interface consists implicitly in grafting a sub-tree to a main tree branch. Thus, branches are dynamically grafted, cut or just temporarily inactivated to represent the different graphical states of the interface.

Like with scalar graphic drawing tools, the different layers of the interface depend on the graphical position into the tree toward the depth-first traversal order (cf. Fig. 1). The first graphical objects, in depth-first traversal order are painted first, designing the “background” of the application. The last ones are painted in front of every other. This order can be altered.

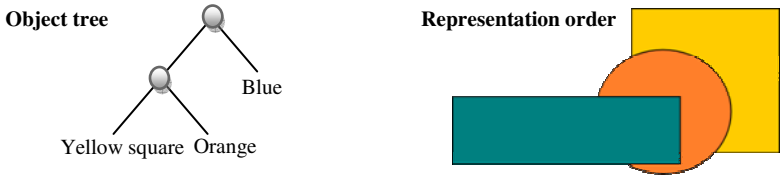


Fig. 1. Layers organization

Simple or complex clips are specified by additional trees. Thus, clips may contain a simple primitive shape so as to a shape composition base (cf. fig. 2). In the same way, texture pattern can be generated from a tree representing the texture pattern (they also can be loaded from an image file).

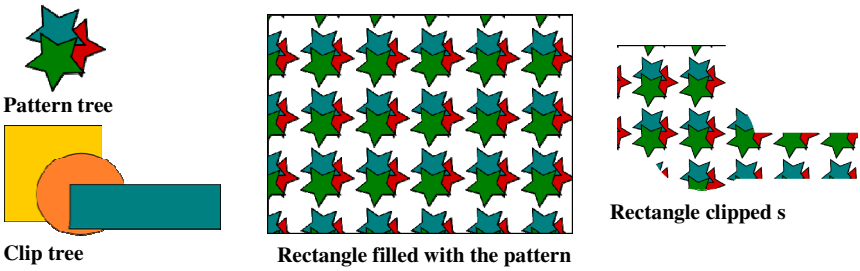


Fig. 2. Using graphic sub-tree as pattern or clip

2.2 Basic Interactions

Tree nodes can handle single and multiple pointer interactions. When e pointer interaction occurred, the toolkit checks if the interaction was performed into the shapes represented by the sub-tree of the handling node (the process consider the specific – and sometime complex – geometry of the shape). Naturally, the evaluation of handled nodes is performed in the inverse depth-first traversal order, corresponding to the order of graphical objects visualized by the user. The events intercepted by a node are propagated to the node ancestors.

Tree node can also handle key events. As standard toolkits, IntNovate manages a focused node, implicitly auto-determined by the last performed click and eventually altered by an explicit request.

At last, tree nodes can handle java and node drag and drop events.

2.3 Graphic Integration

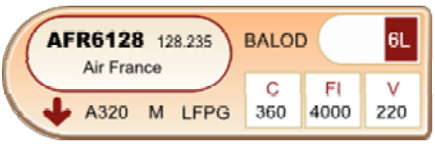
The global structure of IntNovate application, such as the other graphical characteristics (like strokes properties or gradients), is very closed to the SVG and other scalar graphic format definitions.

Because graphic is easier elaborated and altered using drawing tools (graphic designer tools) than described by geometric path or painting directives, IntNovate enables to load graphical sub-trees from scalar graphic files. SVG tree or sub-trees (identified by a node path or node name) are parsed and mapped into IntNovate nodes.

The use of external scalar graphic files enables to ease the interaction between graphic designers and developers. Alterations done by graphic designers are automatically incorporated into the application. Moreover, at the differences with simple

images, the loaded graphic objects may be dynamically altered (for instance: colors, transformation and position, or text values, cf. Fig.3).

a) SVG Model



c) Instance

b) The model is loaded and graphic characteristic dynamically altered

```
IGroup strip=instance
      ("strip.svg#strip");
((IText)strip.find
("callsign"))
          .setText ("AFR6128");
((IText)strip.find
```

Fig. 3. Dynamic alteration of "images"

In order to increase the loading performances, for every loaded SVG tree or sub-tree IntNovate generates a java object. The object constructor instances the sub-tree nodes. The java class is dynamically compiled and loaded. If the object has ever been created and the graphical files is unchanged, IntNovate reuse the compiled object.

The process enables to speed-up the graphical component loading by skipping the parsing step when the graphical file is unchanged (what is the normal software exploitation situation). But, it also enables to protect the graphic designer work. Thus, at the end the development, the graphic source files may be removed from the project. The graphic design is delivered as java compiled classes.

2.4 Cache

Some branches of the tree may contain several and complex graphical objects. The sub-tree rendering time may be long, prejudicing the number of frames rendered per second. To improve the rendering of these sub-trees, IntNovate enables to create image cache for them. When the application tree is redrawn, if a node of a cached sub-tree has been changes, the cache image is rebuilt; else, the cached image is only rendered.

The cached nodes can be specified by the developers. But, IntNovate can also auto-determines the candidate branches. To evaluate the candidate branches, the toolkit measure the sub-trees drawing time, but also use heuristic such as interaction handled by the nodes, animated nodes, user class of components and sub-trees loaded from scalar graphic files. The aim of this evaluation is to detect the sub-trees that are complex to draw, but also the group of graphical object that make part of the same semantic group of object and which shape and mutual position remain unchanged the most part of the time (for instance background, items of a list, animated object, etc. cf. Fig. 4).

Example: application for air traffic control

a) The background contains several graphical objects that may be configured by the user in function of the context. The most part of the time they remain unchanged. The background may be cached.

b) The flight comets seam to be integrated to the background (at the front), whereas their position change all the time. So they won't be

c) Frequent changes in the right branch ➔ group not cached

a) no frequent changes between objects (lines, levels, etc.) at the background ➔ group

b) Flight comets: frequent changes between object position➔ group not

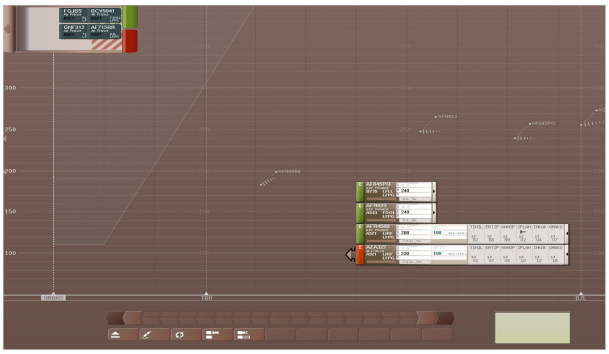


Fig. 4. Principal of cache images

2.5 Widgets

Even if the toolkit is not specifically designed to create form based applications, it proposes a set of skinable widgets containing the principal widgets such as simple, multiline and masked edits, toggle and buttons, radio-buttons, checkboxes, combo-boxes, spinner, list, tables, trees, scrollbar, scroll-panes and split-panes. The widgets have a default skin, whereas the skin can also be specified in SVG.

2.6 Control of Object Transformation: Resizing Specified by Example

Designing widgets with a graphical object tree instead of painting directives introduces a problem in widget resizing. Widgets are normally designed to be represented at different sizes. But, for instance, increasing the size of 25% for a button does not necessarily mean that we are going to increase the size of the border of 25%, neither the size of the font.

In IntNovate, the shapes and their properties are specified with an initial size. A simple way to transform the shapes to make them fit with the other sizes would be to scale them. However, a simple scaling transforms whole graphic without discriminating the semantic aspects of each shape property. Then, it resizes proportionally every graphical property of every shape (the shape but also the borders, text fonts, relative position between shapes, etc.). Consequently, using a simple scale to increase a button size of 25% would also increase of 25% the border, the font size, the text padding, etc (cf. fig. 5).

Fig. 5. A simple scale of the image transforms every aspect of the image proportionally

In order to specify how to resize the shapes respecting the semantic information of the shape properties, IntNovate enables to describe by example how to resize the shapes. The technique is inspired by Flash animations and Artistic Resizing []. It consists in specifying the component appearance at different sizes (key representations). When the component is resized, the value of every property is calculated by interpolation between the property value of key representations with an inferior and a superior size (cf. fig. 6).

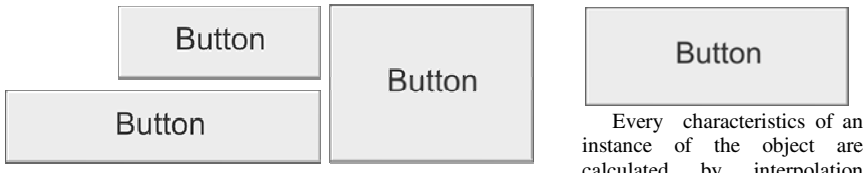


Fig. 6. Transformation by example

2.7 Animation

The toolkit enables to control animations performed on sub-trees. It proposed predefined animations such as scale, translate, opacity animations, and proposes an animation motor enabling the user to specify his proper animations. Pacers (such as fade-in, fade-out, oscillations, etc.) enable to control the animation rhythmic.

Animations may be sequenced and combined through animation scenario models describing: when the different animations should start and end (relatively to the others animations of the scenario). The transformations performed during scale animations may also be specified by example (cf. 2.6).

2.8 Predefined Behaviors

Out of the animation, several predefined behaviors may be applied to sub-trees. Some behaviors, such, as hover-feedbacks, selection feedbacks, or blinking, enable to control active sub-trees toward interactions.

Motion and inertia enable to move an object into a spaced with or without magnetic anchor and space constraints.

Other behaviors enable to control gesture inputs. The toolkit mainly provides a recognition gesture system based on a neuronal network and electronic ink turning objects scriptable.

At last, some behaviors (such as lasso selection, anti-covering) enable to manage interaction between object groups.

2.9 Synthesis

The toolkit is inspired by the Zinc [12] canvas, IntuiKit [1, 2] graphic cooperation design process between designers and programmers, MTools [10] and Flash [17] behaviors, and power-point [16] and Flash animation design. It regroups those different characteristics into a Java environment. J2SE/J2EE, J2ME and Android wrappers enable to reach the different platforms and devices (web, desktop, mobile platforms and windows, linux and Mac OS operational systems).

3 Evaluation

3.1 Design

To evaluate the toolkit usability, 4 J2SE developers compared the development of the same simple application using both IntNovate and only the J2SE JDK.

The subjects were preliminary trained during 4 hours to use IntNovate. The training consists in reproducing short examples illustrating the main skills of the toolkit: graphic integration, layers organizations, animations, and predefined behaviors integration.

The application to develop consisted in (i) dispatching randomly instances of 3 kinds of shapes into the canvas; (ii) provide an interaction to select the same shapes; (iii) create an animation to group the selected shapes; (iv) provide an interaction to input textual information to the group; (v) provide an interaction to organize the groups into a grid. A demonstration of the required application where presented to the subjects.

Graphic design was not the purpose of the evaluation so graphics were provided both as SVG files and as shape paths and text span descriptions. Thus, the resources were prepared to be directly integrated into IntNovate or J2SE. Algorithms required to implement the different behaviors was also provided.

The evaluation was performed during one day. The morning, 2 developers started the development using only the J2SE JDK and the 2 others started using IntNovate. Then, the afternoon, they switched technology.

3.2 Results

The subjects concluded the application development in less than 2 hours and a half the morning and less than 2 hours the afternoon using the IntNovate, and about 4 hours and a half the morning and 3 hour and a half the afternoon using only J2SE paradigms. So, whatever the order of the technology used, the developers reduced significantly the development time using IntNovate.

The written code was about 2 times shorter using the toolkit due to externalization of graphics and to the use of pre-defined behaviors. The 2D scene resulting of the code developed with IntNovate was drawn 15% faster (in frame/s).

The code developed using IntNovate was compatible with a large set of mobile devices, the code develop with J2SE was not because it was depending on J2SE Graphics2D functions different with J2ME Graphic ones and Android ones.

The user developers expressed that the toolkit was easy to use and powerful.

4 Conclusion

We proposed a toolkit, IntNovate, enabling to create rich graphic and interactive applications. The toolkit developed in Java ensure the compatibility with a large set of devices and platforms (mobile, web and desktop). The graphic integration eases the cooperation between graphic designer and developers. The toolkit also aggregates several predefined algorithm for HMI and behaviors turning the development rapid and simple.

A first evaluation illustrates the toolkit efficiency. It is the same for its usage in research projects.

References

1. Chatty, S., Sire, S., Vinot, J.L., Lecoanet, P., Lemort, A., Mertz, C.: Revisiting visual interface programming: creating GUI tools for designers and programmers. In: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, UIST 2004, Santa Fe, NM, USA, October 24-27 (2004)
2. Chatty, S., Sire, S., Lemort, A.: Vers des outils pour les équipes de conception d'interfaces post-WIMP. In: Proceedings of IHM 2004, Namur, Belgium, pp. 45–52. ACM Press (2004)
3. Bier, E.A., Stone, M.C., Pier, K., Buxton, W., Derose, T.D.: Toolglass and magic lenses: the see through interface. In: Proc. ACM SIGGRAPH 1993, pp. 73–80. ACM Press (1993)
4. Raisamo, R., Räihä, K.: A new direct manipulation technique for aligning objects in drawing programs. In: Proc. ACM Symposium on User interface Software and Technology, UIST 1996, pp. 157–164. ACM Press (1996)
5. Rubine, D.: Specifying gestures by example. In: Proc. ACM Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991, pp. 329–337. ACM Press (1991)
6. Dragicevic, P.: Combining Crossing-Based and Paper-Based Interaction Paradigms for Dragging and Dropping Between Overlapping Windows. In: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST 2004), pp. 193–196. ACM Press (2004)
7. Dragicevic, P., Chatty, S., Thevenin, D., Vinot, J.L.: Artistic resizing: a technique for rich scale-sensitive vector graphics. In: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology, Seattle, WA, USA (2005)
8. Kurtenbach, G., Buxton, W.: Issues in combining marking and direct manipulation techniques. In: Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology, UIST 1991, Hilton Head, South Carolina, United States, November 11-13, pp. 137–144. ACM, New York (1991)

9. Merlin, B., Hurter, C., Benhacène, R.: A solution to interface evolution issues: the multilayer interface. In: Proceeding of CHI 2008, Florence, Italie (2008)
10. Merlin, B., Hurter, C., Raynal, M.: Bridging software evolution's gap: The multilayer concept. In: Kurosu, M. (ed.) HCD 2009. LNCS, vol. 5619, pp. 266–275. Springer, Heidelberg (2009)
11. Merlin, B.: Conception et évaluation des claviers logiciels, Méthodologie et instrumentalisation, Éditions Universitaires Européennes, Berlin (2012)
12. Mertz, C., Chatty, S., Vinot, J.L.: The influence of design techniques on user interfaces: the DigiStrips experiment for air traffic control. In: HCI Aero (September 2000)
13. Shneiderman, B.: The future of interactive systems and the emergence of direct manipulation. In: Proceedings of the NYU Symposium on User Interfaces on Human Factors and Interactive Computer Systems, Norwood, NJ, USA, pp. 1–28. Ablex Publishing Corp. (1984)
14. http://www.itproportal.com/2012/07/10/one-size-fits-all-convergence-desktop-and-mobile-operating-systems/
15. http://www.pcworld.com/article/2047067/how-windows-os-x-and-ubuntu-are-slowly-turning-your-pc-into-a-smartphone.html
16. http://office.microsoft.com/pt-br/powerpoint/
17. http://www.adobe.com/devnet/actionscript.html