Signifidgets : What you see is what widget !

Géry Casiez, Sylvain Malacria, Eva Mackamul

Signifidgets : What you see is what widget !

ou comment adapter la représentation des widgets aux interactions qu'ils supportent

Géry Casiez
Univ. Lille, CNRS, Inria, UMR 9189 CRIStAL
Lille, France
gery.casiez@univ-lille.fr

Sylvain Malacria
Univ. Lille, Inria, CNRS,UMR 9189 CRIStAL
Lille, France
sylvain.malacria@inria.fr

Eva Mackamul
Univ. Lille, Inria, CNRS, UMR 9189 CRIStAL
Lille, France
eva.mackamul@inria.fr

ABSTRACT

GUIs are composed of different interactive graphical components, or widgets, with which users can interact to specify their intentions to the system. These widgets are diverse, ranging from simple selectable buttons to range sliders. They have a pre-defined representation and react by default to a list of basic user inputs. Existing programming toolkits use a strong coupling between a widget, its appearance, and the user inputs it supports. Signifidgets are a reflection on how programming APIs could (and should?) approach the use of widgets, allowing programmers to add different possible user inputs to a component, and modify its behavior and appearance to signify and react to them.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; **Graphical user interfaces**; **User interface toolkits**.

IHM'23, April 03–07, 2023, Troyes, France

KEYWORDS

Interaction programming; widgets; signifiers; discoverability.

RÉSUMÉ

Les interfaces graphiques sont composées de différents composants graphiques interactifs, ou *widgets*, avec lesquels les utilisateurs peuvent interagir pour préciser leurs intentions au système. Ces widgets sont divers, pouvant aller de simples boutons sélectionnables à de doubles potentiomètres. Ils ont un affichage prédéfini et réagissent par défaut à une liste précise d'entrées utilisateurs. Les boîtes à outils existantes pour programmer des interfaces graphiques utilisent un couplage fort entre un widget, son apparence, et les entrées utilisateur qu'il supporte. Les Signifidgets sont une réflexion sur la façon dont les API de programmation pourraient (et devraient ?) aborder l'utilisation des widgets, permettant aux programmeurs d'y ajouter différentes entrées utilisateur possibles pour modifier son comportement, et son apparence dans le but de signifier ces entrées.

MOTS CLÉS

Programmation d'interaction ; composants graphiques ; widgets ; signifieurs ; découvrabilité.

INTRODUCTION

Les Interfaces Graphiques Utilisateurs (GUIs) sont principalement construites via l'assemblage de différents composants graphiques interactifs, ou *widgets*, avec lesquels les utilisateurs peuvent interagir pour spécifier leurs intentions au système. Ces widgets sont divers et variés, allant du simple bouton sélectionnable, au double potentiomètre, en passant par des potentiomètres circulaires.

Aujourd'hui, les API de programmation d'Interfaces Graphiques Utilisateurs intègrent toutes un grand nombre de widgets. Chacun de ces widgets a un affichage prédéfini et réagit par défaut à une liste d'actions élémentaires de l'utilisateur [1–3, 6, 8]. Typiquement, un simple bouton change d'apparence quand il est pressé (mouseDown, touchDown), et reprend son apparence d'origine quand il est relâché (mouseUp, touchRelease). Il envoie des événements à chacun de ses changements d'état, auxquels le développeur peut s'abonner en associant des fonctions de rappel (callbacks). Ces changements de représentation restent subtils. Pour reprendre l'exemple du bouton, rien ne suggère à l'utilisateur la possibilité d'effectuer un appui long sur un bouton ou un double tap pour réaliser des actions particulières, quand elles sont supportées. Ce problème de découvrabilité des interfaces empêche les utilisateurs novices d'accéder à des commandes du système [4, 5].

1- Signifidget par défaut

```
// Instantiate a Signifidget
sgt= Signifidget(canvas, 50, 50)
```

2- Ajout du support click

```
// Set a callback when clicked
def clickCallback():
    log("click")
sgt.addClickInput(clickCallback)
```

3- Ajout du support double-click

```
// Set a callback when doubleclicked
def doubleCallback():
    log("double")
sgt.addDoubleInput(doubleCallback)
```

4- Ajout du support long press

```
// Set a callback when dwelled
def dwellCallback():
    log("dwell")
sgt.addDwellInput(dwellCallback)
```

Figure 1 : Exemple de programmation de Signifidget et son apparence associée. 1- un Signifidget est instancié et apparaît désactivé ; Des callbacks pour supporter les clics (2-), double clics (3-) et appuis longs (4-) sont ajoutées et l'apparence du Signifidget est systématiquement mise à jour pour refléter le support de ces entrées et les suggérer à l'utilisateur.

LES SIGNIFIDGETS

Les boites à outils existantes utilisent un fort couplage entre un composant graphique, les actions utilisateurs qu'il supporte, et l'apparence du composant graphique. L'apparence, préconçue par les designers de l'API de programmation, ayant été choisie pour signaler à la fois les entrées utilisateurs possibles avec ce composant, ainsi que les réactions à ces entrées. Néanmoins, si un développeur souhaite augmenter l'interaction avec ce composant, par exemple pour qu'il puisse réagir à des double-clics ou des appuis longs, alors deux options sont offertes : (1) le développeur utilise un autre composant graphique conçu pour gérer ces entrées supplémentaires, sous réserve qu'un tel composant existe ; (2) le développeur continue à utiliser le même composant graphique et modifie son application pour que les entrées utilisateurs souhaitées soient gérées, ce qui demande un effort de programmation important, en particulier si l'on souhaite modifier l'apparence du composant graphique pour qu'il suggère les nouvelles interactions possibles et y réagisse.

Dans cette démonstration, nous présentons le concept de *Signifidget* (Figure 1), une réflexion sur comment les APIs de programmation pourraient (et devraient ?) approcher l'utilisation de composants graphiques pour que les interactions supportées ne soient plus présupposées et dans le but d'offrir le maximum de flexibilité lors de la programmation de l'interface.

Plus précisément, le programmeur peut ajouter différentes interactions utilisateurs possibles à un *Signifidget* qui en réaction va modifier son comportement et son apparence pour supporter ces interactions quand elles sont effectuées et *signifier* [7] qu'elles sont possibles.

La figure 1 illustre la programmation et la représentation d'un Signifidget pour un bouton. Si un bouton est créé mais ne supporte aucune interaction, il apparaît grisé pour signifier l'absence d'interaction possible. Si le développeur ajoute le support aux clics ou taps sur le bouton, en ajoutant une callback avec la méthode *addClickInput*, sa représentation change pour indiquer que le bouton est activable. L'ajout de callbacks pour le support du "double-clic" ou "double-tap" change à nouveau la représentation du bouton pour signifier à l'utilisateur que les deux types d'interaction sont possibles. Visuellement deux cercles sont ajoutés à côté du bouton et se remplissent à chaque clic pour suggérer la possibilité de cette action à l'utilisateur. Enfin, l'ajout du support "appui long" change à nouveau la représentation du bouton. Cette fois une barre de progression horizontale est ajoutée au-dessus du bouton et se remplit dès que l'utilisateur clique ou appuie sur le bouton, suggérant à l'utilisateur qu'il peut continuer d'appuyer dessus. La démonstration comprend un canvas sur lequel il est possible d'ajouter des boutons et un interpréteur Python qui permet d'écrire le code correspondant.

CONCLUSION

Les Signifidgets proposent d'adapter la représentation graphique des widgets suivant les actions qui sont effectivement supportées. La démonstration proposée permet d'illustrer ces concepts.

RÉFÉRENCES

[1] Apple. 2023. App Development with UIKit. Retrieved retrieved February 10th, 2023 from https://developer.apple.com/documentation/uikit/about_app_development_with_uikit

[2] Stéphane Chatty, Stéphane Sire, Jean-Luc Vinot, Patrick Lecoanet, Alexandre Lemort, and Christophe Mertz. 2004. 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 '04)*. Association for Computing Machinery, New York, NY, USA, 267–276. https://doi.org/10.1145/1029632.1029678

[3] Alexandre Demeure, Sylvie Rouillard, François Bérard, and Gaelle Calvary. 2004. Requis et Pistes Pour Les Futures BoîTes à Outils d'interaction Graphiques. In *Proceedings of the 16th Conference on l'Interaction Homme-Machine (IHM '04)*. Association for Computing Machinery, New York, NY, USA, 211–214. https://doi.org/10.1145/1148613.1148649

[4] Alix Goguey, Sylvain Malacria, and Carl Gutwin. 2018. Improving Discoverability and Expert Performance in Force-Sensitive Text Selection for Touch Devices with Mode Gauges. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3174051

[5] Eva Mackamul. 2022. Improving the Discoverability of Interactions in Interactive Systems. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 57, 5 pages. https://doi.org/10.1145/3491101.3503813

[6] Microsoft. 2023. Windows UI Library. Retrieved retrieved February 10th, 2023 from https://learn.microsoft.com/en-us/windows/apps/winui/?source=recommendations

[7] Donald A. Norman. 2013. *The design of everyday things : Revised and expanded edition.* Basic books.

[8] Qt. 2023. Qt software development framework. Retrieved retrieved February 10th, 2023 from https://www.qt.io/product/framework