

Université  
Nice  
Sophia Antipolis

## Introduction à l'Interaction Homme Machine

Gaëtan Rey  
Gaetan.Rey@unice.fr  
DUT Informatique – Janvier 2015

CC BY NC ND

Université  
Nice  
Sophia Antipolis

## C'est quoi l'IHM ?

Interface / interaction Homme-Machine

**ETUDE DE PHÉNOMÈNES MIS EN JEU DANS  
L'ACCOMPLISSEMENT DE TÂCHES AVEC UN SYSTÈME  
INFORMATIQUE**

Quels types de phénomènes ?

- cognitifs
- matériels
- logiciels
- sociaux

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 2

Université  
Nice  
Sophia Antipolis

## Objectifs

Concevoir et réaliser des systèmes/dispositifs/outils/machine/...

**Utiles**  
En conformité avec les fonctions attendues par l'utilisateur cible

**Désirables**  
En conformité avec les valeurs de l'utilisateur cible

**Utilisables**  
En conformité avec les capacités cognitives, sensori-motrices de l'utilisateur cible : confort, efficacité, sécurité, qualité du produit de la tâche réalisée avec le système

**Contextualisé**  
En conformité avec le contexte d'interaction :  
• plate-forme d'interaction  
• environnement physique et social

Fonctionnalité

Utilisateur

Plateforme

Environnement

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 3

Université  
Nice  
Sophia Antipolis

## Malheureusement aujourd'hui

Trop de systèmes sont inadaptés



Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 4

Université  
Nice  
Sophia Antipolis

## IHM et Génie Logiciel

L'IHM s'inscrit dans un processus de développement du Génie Logiciel

**Objectifs**  
Contexte d'interaction  
Modèle de l'utilisateur  
Modèle de tâche  
Modèle des concepts métier  
Qualité

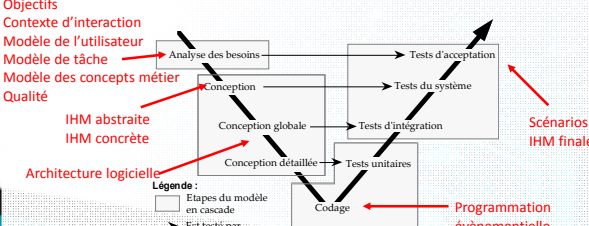
**IHM abstraite**  
**IHM concrète**

**Architecture logicielle**

**Scénarios IHM finale**

**Programmation événementielle**

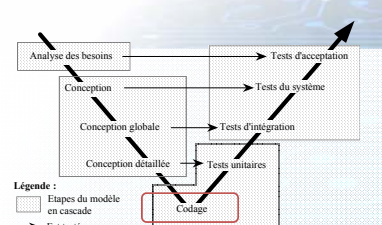
**Légende :**  
□ Etapes du modèle en cascade  
→ Est testé par



Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 5

Université  
Nice  
Sophia Antipolis

## LES INTERFACES ET LA PROGRAMMATION ÉVÉNEMENTIELLE



Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 47

### Les interfaces

#### Un peu de vocabulaire (1)

Labels pointing to components in the 'MainWindow' window:

- Bouton (Button)
- Liste déroulante (ComboBox)
- Bouton Radio (RadioButton)
- Champ de mot de passe (PasswordBox)
- Champ de texte (TextBox)
- Sélecteur de date (DatePicker)
- Zone de liste (ListBox)
- Curseur de défilement (Slider)
- Barre de progression (ProgressBar)
- Etiquette (Label)
- Fenêtre (Window)
- Barre de défilement (ScrollBar)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 48

### Les interfaces

#### Un peu de vocabulaire (2)

Labels pointing to components in the window:

- Onglet (Tab)
- Ruban (Ribbon)
- Menu circulaire (Pie/Radial Menu)
- Menu contextuel (Context/Contextual/Shortcut/Pop-up Menu)
- Cliquez
- Cliquez pour ajouter c

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 49

### Les interfaces

#### Un peu de vocabulaire (3)

Labels pointing to components in the window:

- Barre de titre (Title Bar)
- Iconne (Icon)
- Indicateur d'activité (Throbber)
- Barre d'état (Status Bar/Line)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 50

### Les interfaces

#### Des éléments non visible

- Les conteneurs (Container)
  - Objets qui vont contenir d'autres éléments graphiques
  - Forment une structure hiérarchique (containment hierarchy)
- Les éléments graphiques
  - Managers d'agencement, Gestionnaires d'éléments

```

graph TD
    JFrame --> JPanel
    JPanel --> JTextField
    JPanel --> JTextArea
    JPanel --> JButton
  
```

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 51

### Placement des composants

- Positionnement absolu/fixe
  - Simple pour une interface de taille fixe
  - Très complexe en cas de modifications de la taille ou de changement d'orientation
  - Possibilité d'avoir plusieurs configurations mémorisées
- Gestionnaire de placement/d'agencement
  - Positionne les composants dans le conteneur
  - Recalcule la position et/ou la taille des composants en fonction de règles et de paramètres sur lui-même et sur les composants qu'il contient

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 52

### Gestionnaire de d'agencement


- En .Net (WPF)
  - Ils servent également de conteneurs (double rôle)
  - Grid, Canvas, StackPanel, DockPanel, ...
- En Java (en swing)
  - On parle de « Layout Manager »
  - On doit l'associer à un conteneur
  - FlowLayout, BorderLayout, GridLayout, BoxLayout, ...

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 53



## Exemple Java : BorderLayout

- Est le gestionnaire d'agencement par défaut d'un `ContentPane` (conteneur principal de la `JFrame`)
- Divisé en 5 zones
  - Nord, Sud, Est, Ouest, Center
  - 1 seul composant par zone
- Il consacre tout l'espace du conteneur aux composants.
- Le composant du centre dispose de la place inutilisée par les autres composants



```

this.getContentPane().add(new JButton("CENTER"), BorderLayout.CENTER);

```

Sélectionne le conteneur de la classe courante (celle-ci est une `JFrame`)

Ajoute un composant dans le conteneur

Crée un bouton anonyme avec « CENTER » comme texte


Indique la zone dans laquelle devra être insérer notre bouton

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 54

## Exemple Java : FlowLayout

- Affiche les composants horizontalement
- Passes à la ligne suivante quand il n'y a plus de place sur la ligne
- Ne calcule la position des composants que sur un changement de taille de son conteneur
  - Possibilité de forcer un calcul avec `revalidate()`
- Chaque composant a sa taille préférée
 

```
bouton1.setPreferredSize(new Dimension(100,50));
```



Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 55

## Quelques outils de constructions d'interfaces graphiques (GUI-Builder)

- Java
  - WindowBuilder (Eclipse)
  - Swing GUI Builder (Netbeans)
  - Swing GUI Designer (IntelliJ IDEA)
  - JFormDesigner (Stand-alone, Eclipse, IntelliJ IDEA, Jbuilder, Netbeans, Jdeveloper)
  - Jigloo GUI Builder (Eclipse, WebSphere Studio)
- C++
  - Visual Studio
  - Ultimate++
  - wxDev C++
  - Qt (Stand-alone, Eclipse, Netbeans)
- .Net
  - Visual Studio
  - SharpDevelop
  - MonoDevelop
- GTK+
  - Glade (C, C++, C#, Vala, Java, Perl, Python ...)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 56

## Programmation séquentielle

- Programmation classique / « procédurale »
  - Les instructions s'exécutent dans un ordre donné
  - Les unes après les autres
- Interactions avec l'utilisateur
  - Au fil des instructions du code
  - L'exécution est bloquée attendant l'utilisateur
- Le déroulement est contrôlé par une séquence d'instructions écrites

```


int nombre = 0;
int res = 0;
printf("Entrez une valeur");
res = scanf("%d", &nombre);
if (res == 1)
    printf("Votre valeur est %d", nombre);
else
    // traitement de l'erreur

```

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 57

## Problème du blocage

- Exemple : un lecteur audio
  - Si attendre une action de l'utilisateur bloque le programme :
    - Comment mettre en pause ?
    - Comment changer le volume pendant la lecture ?
- Avoir un mécanisme non bloquant
  - La programmation événementielle



Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 58

## Programmation événementielle

- Paradigme de programmation fondé sur les événements
- Le déroulement est contrôlé par la survenue d'événements (dont les actions de l'utilisateur)
- Principe
  - On s'abonne à des événements
    - après une certaine ressource (Observable)
  - Quand l'événement se produit
    - les abonnés (Observateur) sont avertis

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 59

## Evènements (1)

- ▶ Un événement
  - ▶ un message transmis par un sujet (objet)
  - ▶ pour signaler l'occurrence d'une action ou d'un changement d'état
- ▶ L'action peut être
  - ▶ provoquée par l'intervention de l'utilisateur,
    - ▶ Exemple : un clic de bouton, ...
  - ▶ levée par une autre logique de programme
    - ▶ Exemple : modifier une valeur d'une propriété
- ▶ Le sujet (objet) qui déclenche l'évènement est appelé *l'émetteur d'évènements*
- ▶ L'émetteur d'évènement ne sait pas quel objet ou méthode va recevoir (gérer) les évènements qu'il lève.

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 60

## Evènements (2)

- ▶ Un processus en 4 étapes
  - ▶ **S'abonner** : indiquer à une entité qu'on est intéressé par telle information
  - ▶ **Attendre** : rien à faire
  - ▶ **Être notifié** : l'entité nous informe qu'il vient de se passer quelque chose
    - ▶ C'est la réception de l'évènement
  - ▶ **Réagir** : faire une action en fonction de l'information associée à l'évènement

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 61

## Comparaison

Programmation séquentielle/classique	Programmation événementielle
<ul style="list-style-type: none"> <li>▶ Le programmeur écrit la boucle principale</li> <li>▶ Programme               <ul style="list-style-type: none"> <li>point d'entrée</li> <li>initialisations</li> <li>répéter                   <ul style="list-style-type: none"> <li>lire une commande</li> <li>traiter une commande</li> </ul> </li> <li>jusqu'à la commande finir</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>▶ Pas de boucle principale               <ul style="list-style-type: none"> <li>▶ elle est enfouie dans la bibliothèque</li> <li>▶ L'ordre d'exécution dépend des évènements, il n'est pas visible dans le programme</li> </ul> </li> <li>▶ Programme               <ul style="list-style-type: none"> <li>fonctions (réaction aux évènements)</li> <li>point d'entrée</li> <li>initialisations</li> <li>initialisation // enfouie dans la bibliothèque</li> <li>tant que (non fin) {                   <ul style="list-style-type: none"> <li>attendre évènement suivant E</li> <li>traiter évènement E</li> </ul> </li> </ul> </li> </ul>

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 62

## Programmation événementielle

- ▶ Comment cela fonctionne-t-il ?
  - ▶ Les évènements sont placés dans une liste FIFO
  - ▶ La boucle de gestion des évènements prend les évènements dans la liste et les traite
    - ▶ XtAppMainLoop() en c (Xt + Motif)
    - ▶ gtk\_main() en GTK+
    - ▶ Event Dispatch Thread en Java
    - ▶ UI thread en C#
    - ▶ ...

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 63

## XtAppMainLoop() Xt + Motif

- ▶ La gestion des évènements est prise en charge par XT
- ▶ La fonction XtAppMainLoop :
  - ▶ lit le prochain évènement : XtAppNextEvent
  - ▶ l'envoi à la procédure appropriée: XtDispatchEvent
- ▶ Cette fonction utilise le champ window de l'évènement pour chercher une widget qui possède cette fenêtre
  - ▶ Les clients qui ont sélectionné les bons évènements sur les (fenêtres des) widgets les reçoivent, et les traitent
  - ▶ Cette gestion est "automatique" pour les widgets qui ont des *tables de correspondances*
  - ▶ Elle est à un niveau d'abstraction supérieur à la gestion des évènements bruts

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 64

## XtAppMainLoop() Xt + Motif (2)

```

graph TD
    XtAppMainLoop --> XtAppNextEvent
    XtAppNextEvent --> XtDispatchEvent
    XtDispatchEvent --> Widget
    Widget --> Application
    Application --> XtAppMainLoop
  
```

The flowchart illustrates the Motif Programming Model. It starts with XtAppMainLoop, which calls XtAppNextEvent to get the next event. XtAppNextEvent then calls XtDispatchEvent to dispatch the event to the appropriate widget. The widget then calls the application's callback function, which returns control to XtAppMainLoop. The flowchart also shows the relationship between the widget and the application, with the widget being a part of the application's window hierarchy.

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 65



Université Nice Sophia Antipolis

## gtk\_main() GTK+

- La fonction `gtk_main()`
  - Se « connecte » au serveur X
  - Utilise une file d'attente d'événements
- `gtk_main()`
  - attendra les événements venant du serveur X et
  - demandera aux widgets d'émettre les signaux lorsque ces événements surviendront.

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 66

Université Nice Sophia Antipolis

## Event Dispatch Thread Java

- EDT est la Thread dans laquelle l'AWT et Swing font
  - leur affichage
  - leur propagation d'événements.
- Les composants AWT et Swing doivent uniquement être utilisés dans l'EDT
- L'EDT alterne
  - les cycles d'affichage
  - les cycles de propagation d'événements.
- Lors de la gestion des événements ou des composants graphiques, il faut vérifier:
  - Qu'on est dans l'EDT
    - AWT : invoquer la méthode statique `isDispatchThread()` de la classe `java.awt.EventQueue`.
    - Swing : invoquer la méthode statique `isEventDispatchThread()` de la classe `javax.swing.SwingUtilities`
  - Que le composant n'est pas en train de se redessiner

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 67

Université Nice Sophia Antipolis

## Event Dispatch Thread Java

Source : <https://www.clear.rice.edu/comp310/JavaResources/GUI/>

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 68

Université Nice Sophia Antipolis

## Thread d'interface utilisateur C#

- Les applications WPF commencent avec 2 threads
  - un pour gérer le rendu qui s'exécute masqué en arrière-plan (rendering thread)
  - un pour gérer l'interface utilisateur qui reçoit les entrées, gère les événements, peint l'écran et exécute le code de l'application (UI thread)
- Le thread d'interface utilisateur met en file d'attente des éléments de travail à l'intérieur d'un objet appelé un Dispatcher
- Le Dispatcher sélectionne les éléments de travail en fonction de leur priorité et exécute complètement chacun d'eux

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 69

Université Nice Sophia Antipolis

## Patron de conception Observateur/Observable

- Appelé également Observer / Observable
- Met en œuvre plusieurs acteurs
  - le sujet (observable) et les observateurs
- Permet à un ensemble d'objets
  - de s'abonner à un sujet dit observable
  - pour recevoir des notifications quand ce sujet change
- Objectif
  - augmenter la réutilisabilité en diminuant le couplage entre les classes
    - le widget bouton peut être utilisé dans plusieurs projets. Il n'y a pas de lien entre le widget et les fonctions appelées dans chaque projet

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 70

Université Nice Sophia Antipolis

## Patron de conception Observateur/Observable

La notion d'observateur/observable permet de coupler des modules de façon à réduire les dépendances aux seuls phénomènes observés

(1) Savoir Observer un sujet observable

(2) Savoir réagir à une notification

(3) Savoir écrire un sujet observable

[Wikipédia]

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 71

## Abonnement et réaction en C (Xt + Motif) (1)

- ▶ L'émission d'un événement va entraîner l'exécution d'une ou plusieurs fonctions
  - ▶ Les « callback »
- ▶ Les fonctions de callback possèdent trois paramètres :
  - ▶ Un pointeur sur le composant (Widget) associé à ce callback.
  - ▶ Le second paramètre sert à passer des données à la fonction de callback
  - ▶ Un pointeur sur une structure qui contient
    - ▶ tous les renseignements relatifs à l'événement qui a provoqué l'appel de la fonction de callback,
    - ▶ des renseignements relatifs au type de Widget passé en 1<sup>er</sup> paramètre,

[Cours Xt + Motif] (Les signaux)

## Abonnement et réaction en C (Xt + Motif) (2)

```
// Définie une fonction de callback
void MaFonction(Widget w, XtPointer clid, XtPointer cad) {
    // Mettre ici le code de la fonction
}

int main() {
    Widget boite, bouton;
    ...
    // Crée un bouton avec boite comme parent
    bouton = XmCreatePushButton(boite, "bouton", NULL, 0);
    // Rend le bouton visible
    XtManageChild(bouton);
    // Associe un callback -> appelez MaFonction quand on « active » le bouton
    XtAddCallback(bouton, XmNactivateCallback, MaFonction, NULL);
    // Lance la boucle de gestion des événements
    XtAppMainLoop(app);
    // Attention, rien n'est exécuté après cette ligne !
}
```

## Abonnement et réaction en C (GTK+) (1)

- ▶ En GTK+, un signal est une méthode qui permet de recevoir de notification
  - ▶ un signal est un événement qui se produit sur un objet
- ▶ Ce sont les signaux qui permettent à l'utilisateur d'une application d'interagir avec l'application.
  - ▶ Par exemple, si un utilisateur clique sur un bouton, un signal d'un objet sera émis dans l'objet spécifié.
- ▶ La fonction spécifiée lors de l'émission de ce signal, sera appelée
- ▶ Un signal est sélectionné en utilisant un identificateur entier (l'id du signal) ou une chaîne de caractères
- ▶ La fonction qui sera appelée lors de l'émission du signal, est appelée fonction callback

[Cours GTK+] (Les signaux)

## Abonnement et réaction en C (GTK+) (2)

```
void OnDestroy(GtkWidget *pWidget, gpointer pData){
    /* Arrêt de la boucle evenementielle */
    gtk_main_quit();
}

int main(int argc, char **argv){
    /* Déclaration du widget */
    GtkWidget *pWindow;
    gtk_init(&argc, &argv);
    /* Création de la fenêtre */
    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    /* Connexion du signal "destroy" */
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(OnDestroy), NULL);
    /* Affichage de la fenêtre */
    gtk_widget_show(pWindow);
    /* Démarrage de la boucle evenementielle */
    gtk_main();
    return EXIT_SUCCESS;
}
```

## Abonnement et réaction en Java (1)

- ▶ Dans le contexte d'une interface graphique (Swing, AWT, ...), les listeners permettent au programmeur de réagir suite aux actions de l'utilisateur
- ▶ Les « listeners » sont des interfaces
  - ▶ fournissent une ou plusieurs méthodes qui peuvent être implémentées différemment selon les cas et les besoins, pour répondre aux événements
- ▶ Chaque listener dispose d'une classe Event associée. Cette classe
  - ▶ étend java.awt.event.EventObject
  - ▶ fournit une description de l'événement capturé.

[Developpez.com]

## Abonnement et réaction en Java (2)

```
// Etape 1 : déclaration de la classe
public class MaClasse {
    // Etape 2 : création d'un bouton.
    JButton monBouton = new JButton("Mon Bouton...");

    /* Etape 3 : création de la classe anonyme */
    ActionListener listener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Bouton a été cliqué
            // ...
        }
    };

    monBouton.addActionListener(listener);
}
```

[Developpez.com]



## Abonnement et réaction en Java (3)

```
// Étape 1 : déclaration de la classe
public class MaClasse {
    // Étape 2 : création de deux boutons.
    JButton monBouton = new JButton("Mon Bouton...");
    JButton monBouton2 = new JButton("Mon Bouton 2...");
    /* Étape 3 : création de la classe anonyme */
    monBouton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Cette méthode ne sera appelée que pour les événements sur le bouton monBouton.
        }
    });
    /* On refait la même chose pour le deuxième bouton */
    monBouton2.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            // Cette méthode ne sera appelée que pour les événements sur le bouton monBouton2.
        }
    });
}
```

[\[developpez.com\]](http://developpez.com)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 78

## Abonnement et réaction en Java (4)

```
// Étape 1 : déclaration de la classe
public class MaClasse implements ActionListener {
    // Étape 2 : Création de deux boutons
    JButton monBouton = new JButton("Mon Bouton");
    JButton monBouton2 = new JButton("Mon Bouton2");

    public MaClasse() {
        // Étape 4 : On ajoute « l'écouteur » sur le bouton « monBouton »
        monBouton.addActionListener(this);
        // Puis sur monBouton2
        monBouton2.addActionListener(this);
    }

    /* Étape 3 : Cette méthode est déclarée dans l'interface ActionListener. Il nous faut l'implémenter. */
    public void actionPerformed(ActionEvent e) {
        // Étape 2bis
        if (e.getSource() == monBouton) {
            // Bouton 1 a été cliqué
        } else {
            // Bouton 2 a été cliqué
        }
    }
}
```

[\[developpez.com\]](http://developpez.com)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 79

## Abonnement et réaction en Java (5)

```
// Fichier : MonListener.java
public class MonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == monBouton) {
            // Bouton 1 a été cliqué
        } else {
            // Bouton 2 a été cliqué
        }
    }
}

// Fichier : MaClasse.java
public class MaClasse {
    JButton monBouton = new JButton("Mon Bouton");
    JButton monBouton2 = new JButton("Mon Bouton2");

    public MaClasse() {
        monBouton.addActionListener(new MonListener());
        monBouton2.addActionListener(new MonListener());
    }
}
```

[\[developpez.com\]](http://developpez.com)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 80

## Abonnement et réaction en C# (1)

- Utilisation de délégués pour la gestion d'événements
- Un délégué
  - Est un type qui encapsule (contient une référence à) une méthode
  - Est déclaré avec une signature
    - qui indique le type de retour et les paramètres des méthodes qu'il référence,
    - et peut contenir des références à des méthodes qui correspondent à la signature.
  - Est l'équivalent d'un appel de fonction ou d'un pointeur de fonction de type sécurisé.
  - Une déclaration delegate suffit pour définir une classe déléguée
  - Une fois initialisé avec une méthode, il se comporte exactement comme cette méthode et peut être appelé avec l'opérateur ()

```
public delegate void SeuilAtteintEventHandler(SeuilAtteintEventArgs e);
```

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 81

## Abonnement et réaction en C# (2)

```
// On indique le type de delegate défini pour l'évènement (EventHandler)
monObjet.MonEvenement += new EventHandler(monObjet_MonEvenement);
/* La signature (type de retour et types des paramètres) de la méthode utilisée pour s'abonner doit correspondre à la signature de ce delegate */
private void monObjet_MonEvenement(object sender, EventArgs e)
{
    Console.WriteLine("MonEvenement s'est produit !");
}

// Depuis C# 2, il n'est plus nécessaire de spécifier le type de delegate
monObjet.MonEvenement += monObjet_MonEvenement;

// Pour se désabonner d'un évènement
monObjet.MonEvenement -= monObjet_MonEvenement;
```

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 82

## Problème du Multi-thread

- Que faire
  - Si j'ai un traitement de mon évènement qui est trop long et qui bloque ma GUI ?
    - Utiliser un autre thread !
  - Si je veux modifier ma GUI depuis un autre thread ?
    - On ne peut pas !
    - Mais on peut contourner le problème
      - Demander au thread qui gère la GUI de faire le travail
        - invokeLater en Java
        - Invoke ou BeginInvoke en C#

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 83

## Gestion de XtAppMainLoop() en Xt + Motif

- ▶ Motif est étroitement couplé avec la X Toolkit
- ▶ Les interfaces X11R6 pour le multi-threading ne sont nécessaires que si l'application multithread appelle les interfaces Motif / Xt dans plusieurs threads
- ▶ Il est possible d'écrire des applications multi-thread Motif dans lesquelles les interfaces Motif / Xt sont invoquées qu'à partir d'un seul fil.
  - ▶ Dans de telles applications, les interfaces ne sont pas nécessaire

[\[Structure of a Motif Program\]](#)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 84

## Gestion de gtk\_main() en GTK+

- ▶ Une réponse simple
  - ▶ Dans le cas d'une gestion multi-thread, vous ne devez appeler la bibliothèque GTK seulement depuis le thread exécutant gtk\_main()
- ▶ En réalité, c'est plus compliqué
  - ▶ [\[Multi-threaded GTK applications – Part 1: Misconceptions\]](#)
  - ▶ [\[Multi-threaded GTK applications – Part 2: java-gnome\]](#)

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 85

## Gestion de l'EDT en Java

- ▶ Exécuter un traitement plus tard dans l'EDT (ou depuis un autre Thread)
 

```
// Cette méthode retourne immédiatement.
SwingUtilities.invokeLater(new Runnable() {
    /**
     * {@inheritDoc}
     */
    @Override public void run() {
        myButton.setText("Salut le monde !");
    }
});
```
- ▶ Lancer un traitement long
  - ▶ Doivent être lancés hors de l'EDT
  - ▶ Pour Java 6+, utilisez la classe javax.swing.SwingWorker qui offre un framework permettant d'exécuter une tâche hors de l'EDT tout en récupérant ses résultats intermédiaires et finaux dans l'EDT.

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 86

## Gestion de l'UI Thread en C# (1)

<pre>// La mauvaise façon de faire  // Travail effectué depuis un autre thread ThreadStart start = delegate() {     // ...  // Ceci provoquera une exception statusText.Text = "Depuis un autre Thread"; };  // Crée et démarre un thread new Thread(start).Start();</pre>	<pre>// La bonne façon de faire (appel synchrone)  // Travail effectué depuis un autre thread ThreadStart start = delegate() {     // ...  // Met à jour le Texte du TextBlock. // Cette action sera effectuée via le dispatcher Dispatcher.Invoke(DispatcherPriority.Normal, new     Action&lt;string&gt;(&lt;SetStatus&gt;, "Depuis un autre Thread ")); };  // Crée et démarre un thread new Thread(start).Start();</pre> <p><a href="#">[Créez des applications plus réactives avec le répartiteur]</a></p>
--	---

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 87

## Gestion de l'UI Thread en C# (2)

```
// La bonne façon de faire (appel asynchrone)
// Travail effectué depuis un autre thread
ThreadStart start = delegate() {
    // ...

// Met à jour le Texte du TextBlock.
// Cette action sera effectuée via le dispatcher
DispatcherOperation op = Dispatcher.BeginInvoke(DispatcherPriority.Normal, new
    Action<string>(<SetStatus>, "Depuis un autre Thread (Async)");
DispatcherOperationStatus status = op.Status;
while (status != DispatcherOperationStatus.Completed) {
    status = op.Wait(TimeSpan.FromMilliseconds(1000));
    if (status == DispatcherOperationStatus.Aborted) {
        // Alert Someone
    }
}
};

// Crée et démarre un thread
new Thread(start).Start();
```

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 88

## Nos Evènements

- ▶ Le mécanisme d'évènements n'est pas figé
- ▶ On peut créer nos propres évènements
- ▶ On peut émettre des évènements dans nos programmes

Janvier 2015 Gaëtan Rey - Université Nice Sophia Antipolis 89



## Définir des évènements en Java (1)

- Pour créer ses propre évènements
  - Une classe évènement
 

```
public class MyEventClass extends java.util.EventObject {
    //le constructeur de la class
    public MyEventClass(Object source) {
        super(source);
    }
}
```
  - Une Interface
 

```
public interface MyEventClassListener {
    public void handleMyEventClassEvent(EventObject e);
}
```

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 90

## Définir des évènements en Java (2)

- Pour créer ses propre évènements
  - Une source de l'évènement
 

```
public class MyEventSource {
    private List _listeners = new ArrayList();
    public synchronized void addEventListener(MyEventClassListener listener) {
        _listeners.add(listener);
    }
    public synchronized void removeEventListener(MyEventClassListener listener) {
        _listeners.remove(listener);
    }
    // call this method whenever you want to notify the event listeners of the particular event
    private synchronized void fireEvent() {
        MyEventClass event = new MyEventClass(this);
        Iterator i = _listeners.iterator();
        while(i.hasNext()) { ((MyEventClassListener) i.next()).handleMyEventClassEvent(event); }
    }
}
```

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 91

## Définir des évènements en Java (3)

- Pour créer ses propre évènements
  - Une Event Source
 

```
public class MyEventListener implements MyEventClassListener {
    // ... Votre code

    //implement la(les) méthode(s) requise de l'interface
    public void handleMyEventClassEvent(EventObject e) {
        // gérer l'évènement comme bon vous semble
    }
}
```

[\[Creating a Custom Event\]](#)

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 92

## Définir des évènements en C# (1)

- Définir d'un évènement
 

```
class MyClass{
    public event EventHandler MyEvent;
}
public void EventHandler(object sender, EventArgs e)
{
    // ... votre code ...
}
// ... votre code ...
new MyClass().MyEvent += new EventHandler(EventHandler);
```

- Utilisez event dans la signature de votre classe d'évènements
- Spécifiez le type du délégué pour l'évènement

[\[Accesseurs d'évènement\]](#)

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 93

## Déclencher des évènements en C# (2)

- Déclencher un évènement
  - Ajoutez une méthode marquée comme protected et virtual
  - Nommez cette méthode OnEventName / QuandNomEvenement
  - La méthode prend un paramètre de type « objet de données d'évènement »

```
class MyClass {
    public event EventHandler MyEvent;

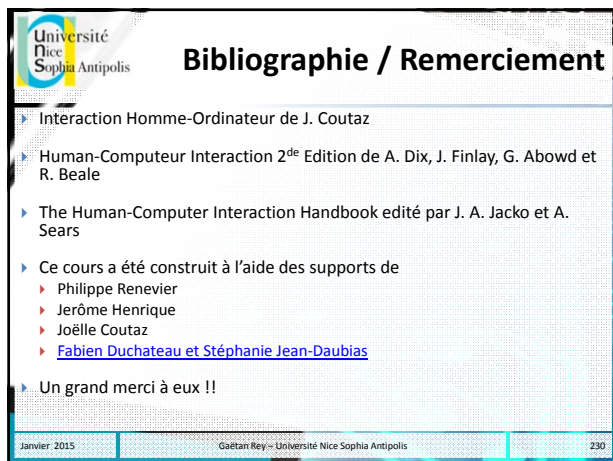
    protected virtual void OnMyEvent(EventArgs e) {
        EventHandler handler = MyEvent;
        if (handler != null) {
            handler(this, e);
        }
    }
    // le reste du code
}
```

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 94

## Le minimum à connaître

- Noms des différents Widgets
- Notions de conteneurs et de gestionnaire d'agencement
- Principe de le programmation évènementielle
- Notion d'évènements
- Patron de conception « observer/observable »
- Notion de Callback, listener, delegate

Janvier 2015 Gaëtan Rey – Université Nice Sophia Antipolis 95



**Bibliographie / Remerciement**

- ▶ Interaction Homme-Ordinateur de J. Coutaz
- ▶ Human-Computer Interaction 2<sup>de</sup> Edition de A. Dix, J. Finlay, G. Abowd et R. Beale
- ▶ The Human-Computer Interaction Handbook édité par J. A. Jacko et A. Sears
- ▶ Ce cours a été construit à l'aide des supports de
  - ▶ Philippe Renevier
  - ▶ Jérôme Henrique
  - ▶ Joëlle Coutaz
  - ▶ [Fabien Duchateau et Stéphanie Jean-Daubias](#)
- ▶ Un grand merci à eux !!

Janvier 2015      Gaëtan Rey - Université Nice Sophia Antipolis      230