Implémentation de systèmes interactifs

Modélisation et programmation par événement

Objectifs

- Première partie
  - Maîtriser le concept de système interactif
  - Maîtriser une technique de description formelle
  - Comprendre le passage du prototype à l'implémentation
  - Maîtriser la notion de composant
- Seconde partie
  - Maîtriser une bibliothèque de développement Front (React js)

Organisation

- Il y a des cours et des TD
- Pour le reste, on verra

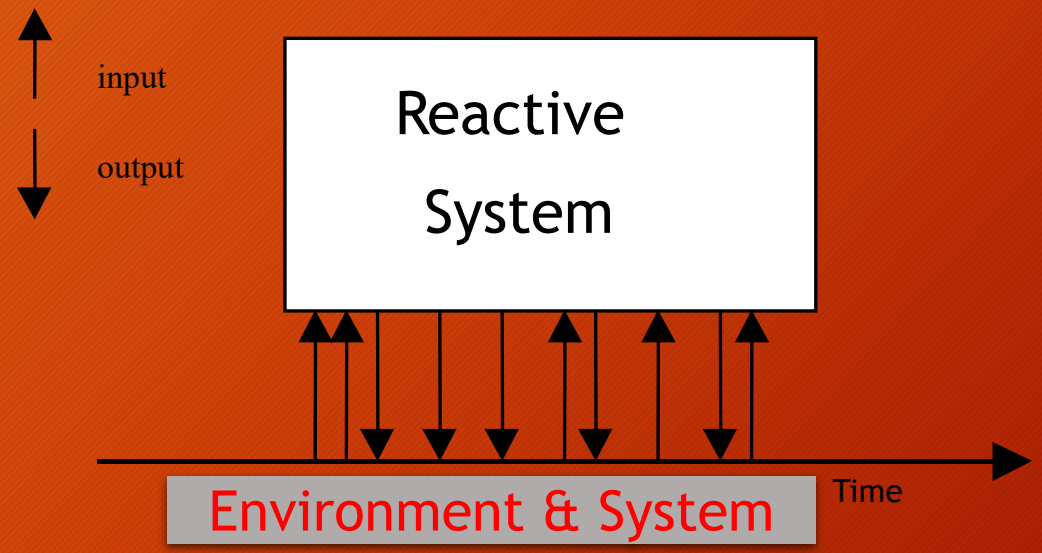
Evaluation

- Un contrôle écrit portant sur la modélisation
- Un TP noté sur React
- En fonction de l’avancée, un TP plus conséquent

Qu’est-ce qu’un système interactif ?

Definition des SI

- Un principe: Des systèmes réactifs
- Un challenge: variabilité (contextes d'utilisation, des utilisateurs, …)
- Une philosophie: Outil comme approche (utilisateur dans la boucle « IN THE LOOP »)



Définition des SI

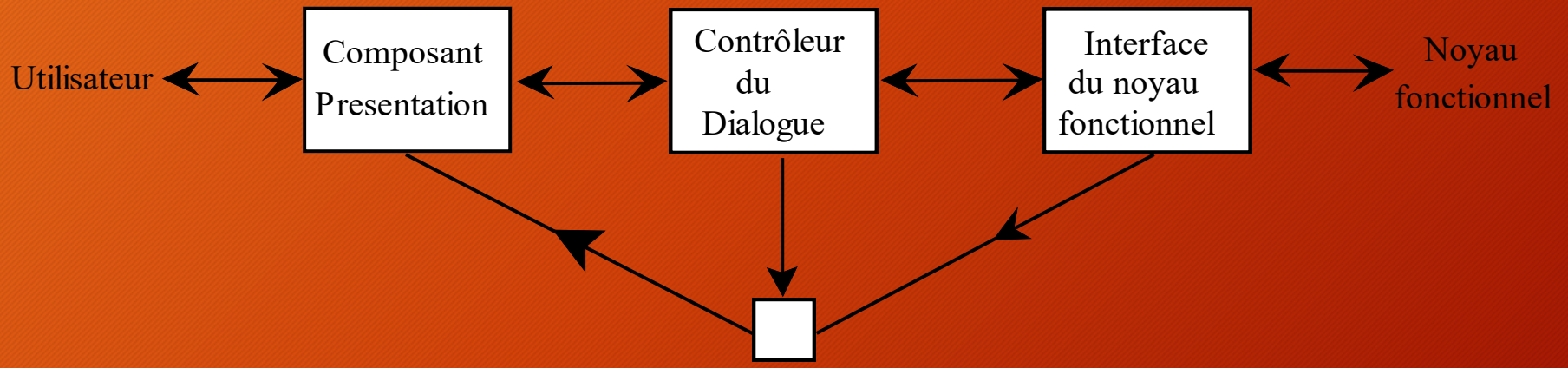
Output

Input

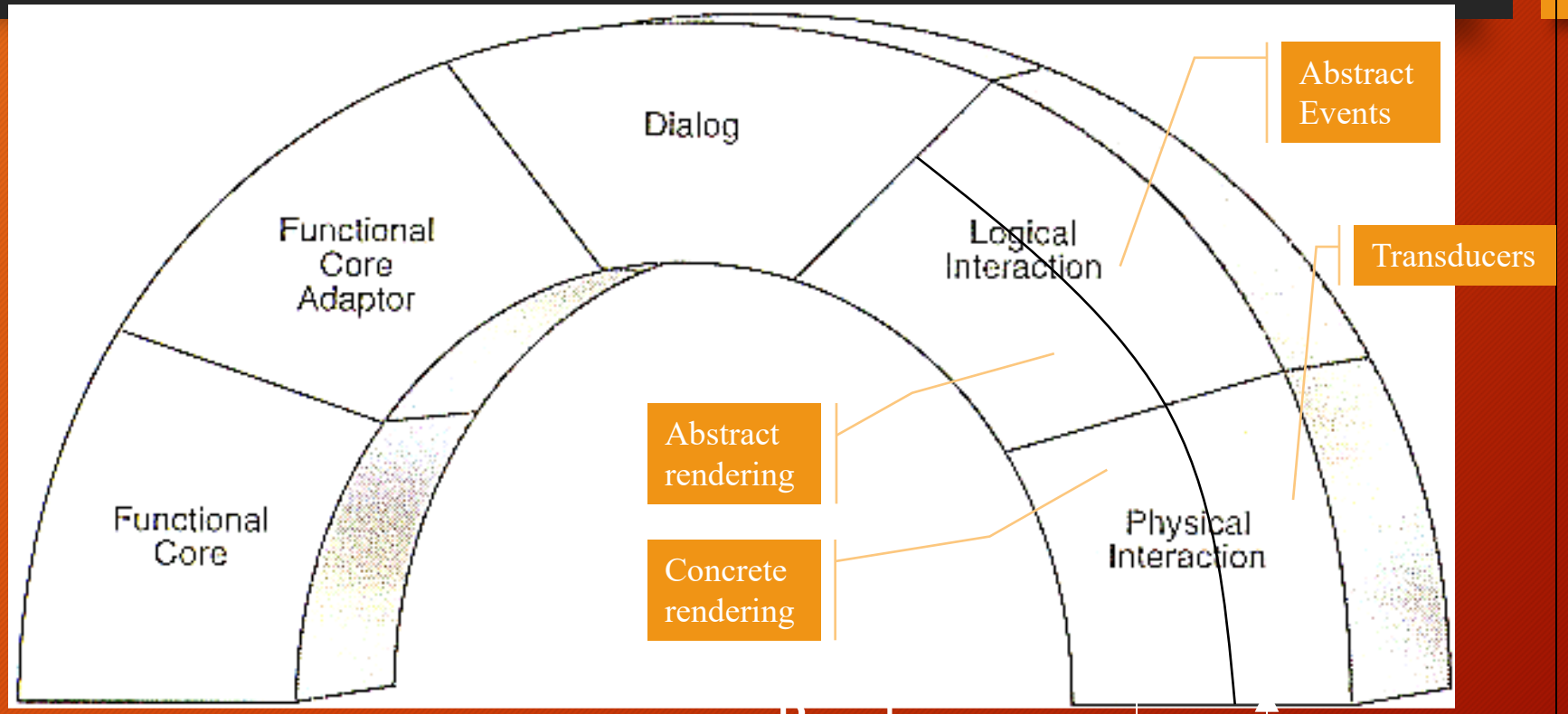
Pnueli A. (1986) Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In: de Bakker J.W., de Roever W.P., Rozenberg G. (eds) Current Trends in Concurrency. Lecture Notes in Computer Science, vol 224. Springer, Berlin, Heidelberg

Du Design à la construction

Modèle de Seeheim 1983



Modèle ARCH 1991



Conception des IHM

- Il faut concevoir les trois parties du modèle de Seeheim:

- La présentation
  - ce que l'utilisateur voit de l'application
- Le dialogue :
  - qu'est-ce que l'utilisateur a la possibilité de faire
  - comment l'utilisateur agit sur la présentation
  - l'influence de son action sur ce qu'il pourra faire ensuite
- Le noyau fonctionnel :
  - les fonctions réalisées par l'application
  - les données manipulées par l'application

Fonctionnement des systèmes par événements

Interaction dirigée par le système

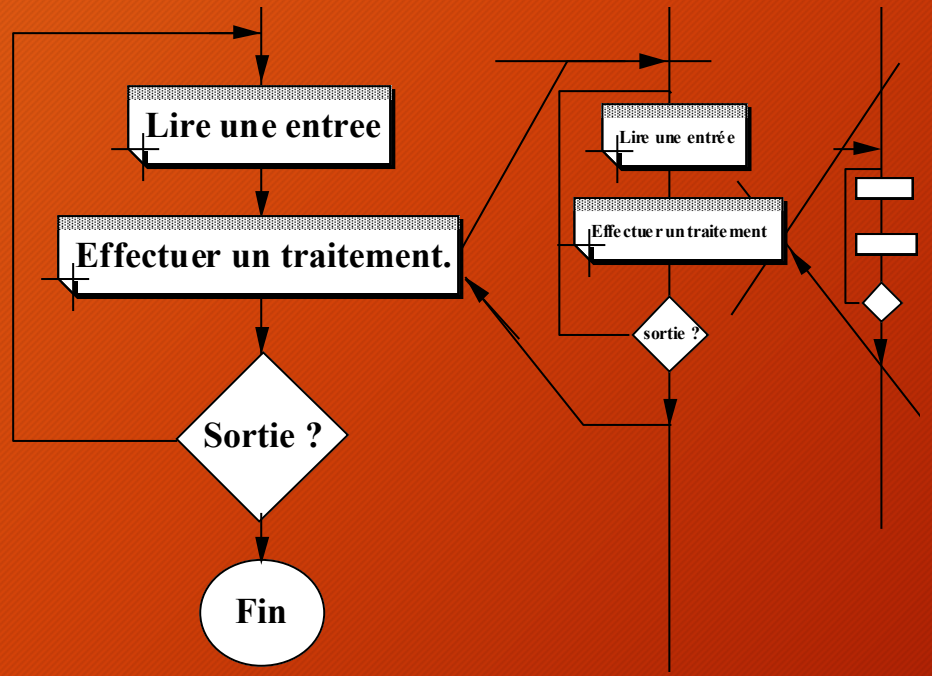
Système en attente d'une entrée utilisateur

```
Début
choix = '1';
Tantque choix <> '9' faire
   affiche-menu;
   lire(choix);
   case choix of
   1 : ajouter;
   2 : modifier;
   3 : supprimer;
   9 : Quitter;
   Fin Case
FintTantque
Fin
```

Utilisateur en attente du calcul

```
Procédure Ajouter;
début
rep = 'o';
Tantque rep <> 'n';
   dessin-écran;
   lire(nom);
   lire(prenom);
   ...
   écrire('voulez-vous continuer ?');
   lire(rep)
FinTantque
Fin
```

Vision algorithmique du monde

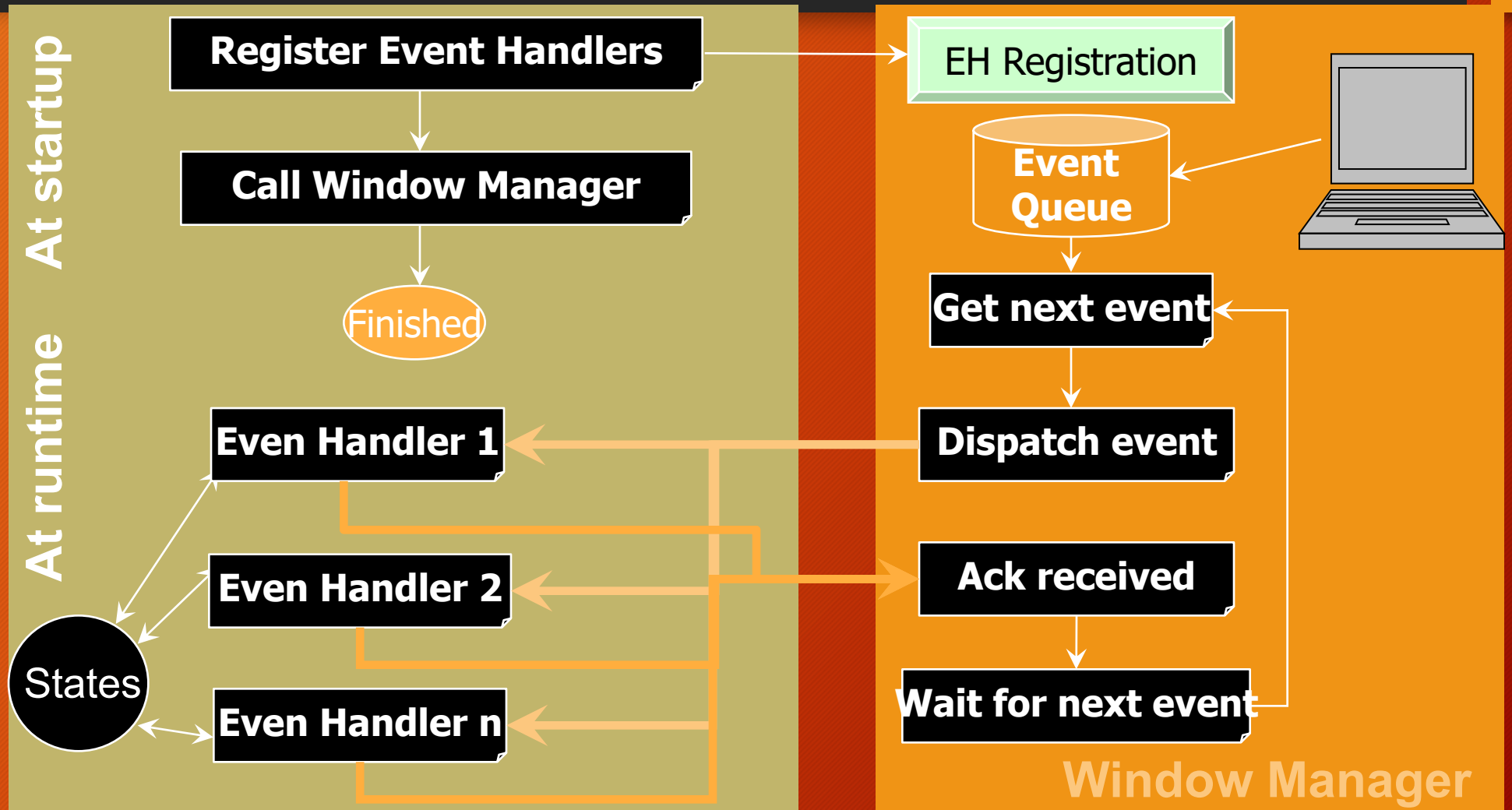


Structure application par événement

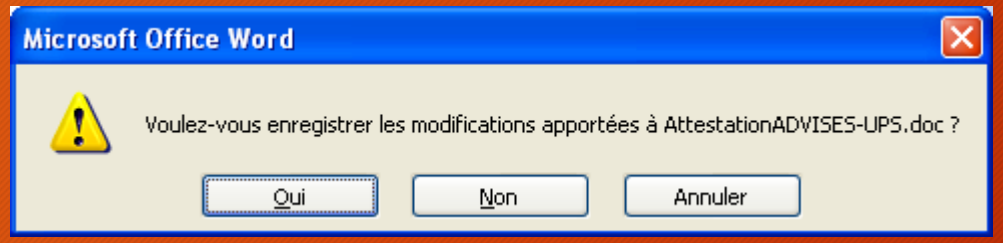
- La boucle d'événement (main event loop)
  - reçoit chaque événement produit par l'utilisateur
- Les gestionnaires d'événements : sont des procédures associées à chaque couple (widget, action sur un widget) et appelées par la main event loop dès que une action a été réalisée.

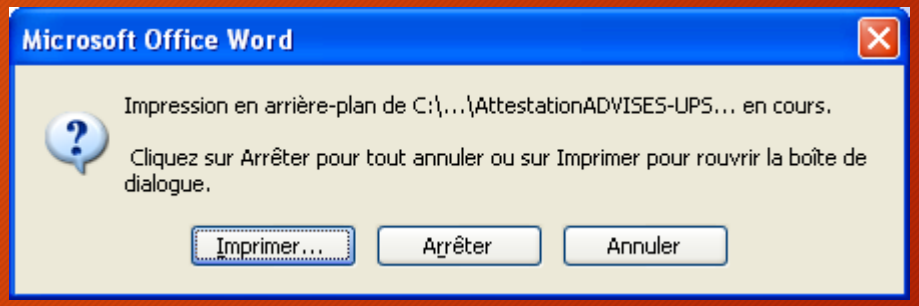
- Tous les event handlers ont la même structure
  - EventHandler1;
  - Précondition;
  - Action;
  - Modification de l'état du dialogue;
  - Rétroaction graphique;
  - Fin EventHandler1;

Event-based Functioning



Comportement basé sur les états





Deuxième Couche

Principe Télétubbies

Principe Gillette

Exécution typique d'un programme

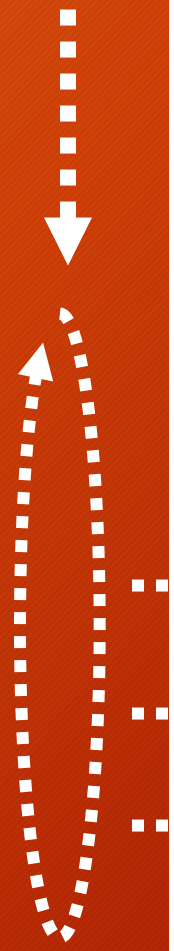
- Non-interactive
- Exécution linéaire
- Processus d'automatisation (automatique)
- Ne prends pas en compte les capacités de l'humain versus ordinateur

program:

```
main()
{
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
}
```

Programme Interactif à choix multiples

- L'utilisateur choisit les options
- Exécution non-linéaire ("branching")
- Ordre imprédictible
- Système arrêté sur instruction de lecture
- Possibilité de continuer



program:

```
main()
{
        decl data storage;
        initialization code;

        loop
        {
                show options;
                read(choice);
                switch(choice)
                {
                        choice1:
                                code;
                        choice2:
                                code;
                        …
                }
        }
}
```

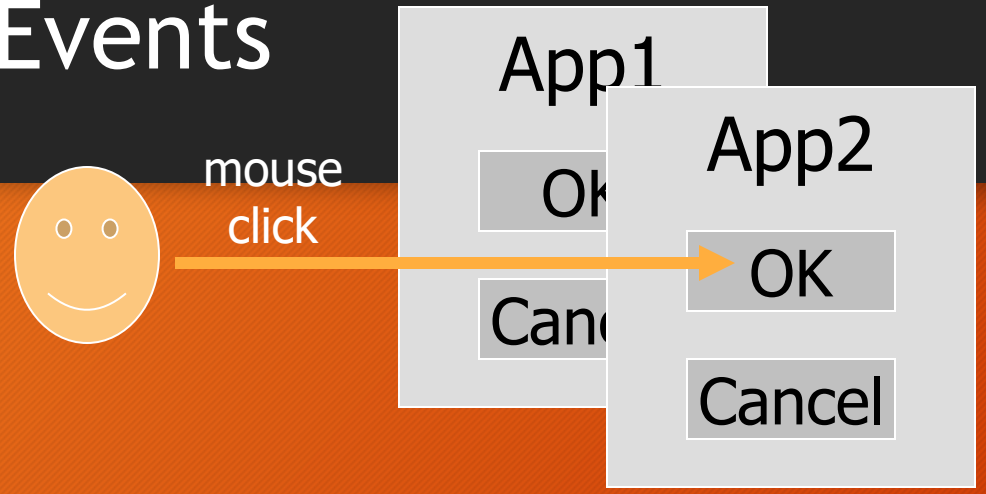
Interface Dirigée par l'utilisateur

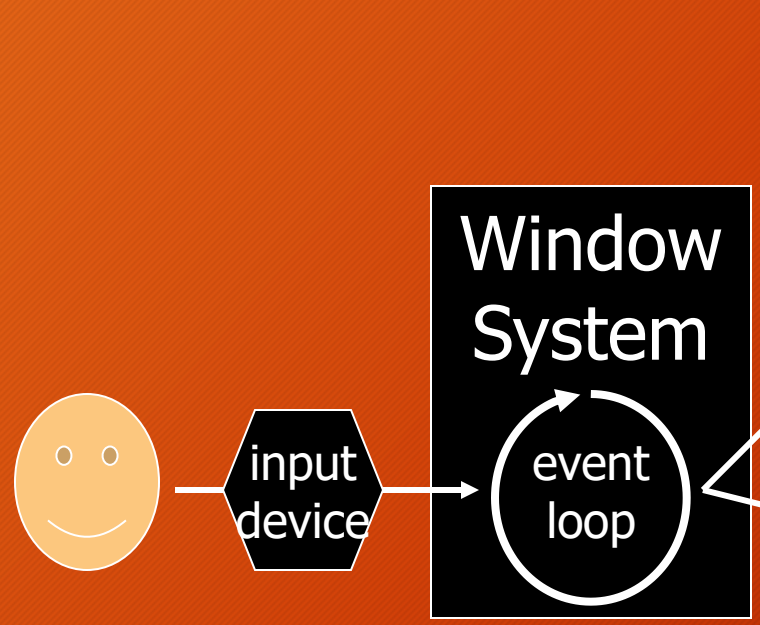
- L'utilisateur déclenche des commandes
- Exécution non linéaire
- Ordre non prédictible
- La plupart du temps le système ne fait rien
- Les procédure de gestion d'événements

GUI program:

```
main()
{
      decl data storage;
      initialization code;

      create GUI;
      register callbacks;

      main event loop;
}

Callback1()    //button1 press
{     code; }
Callback2()    //button2 press
{     code; }
…
```

GUI Events





App2 code:

```
OKbtn_click()
{
    do stuff;
}
CancelBtn_click()
{
    do different stuff;
}
App2Form_click()
{
    do other stuff;
}
```

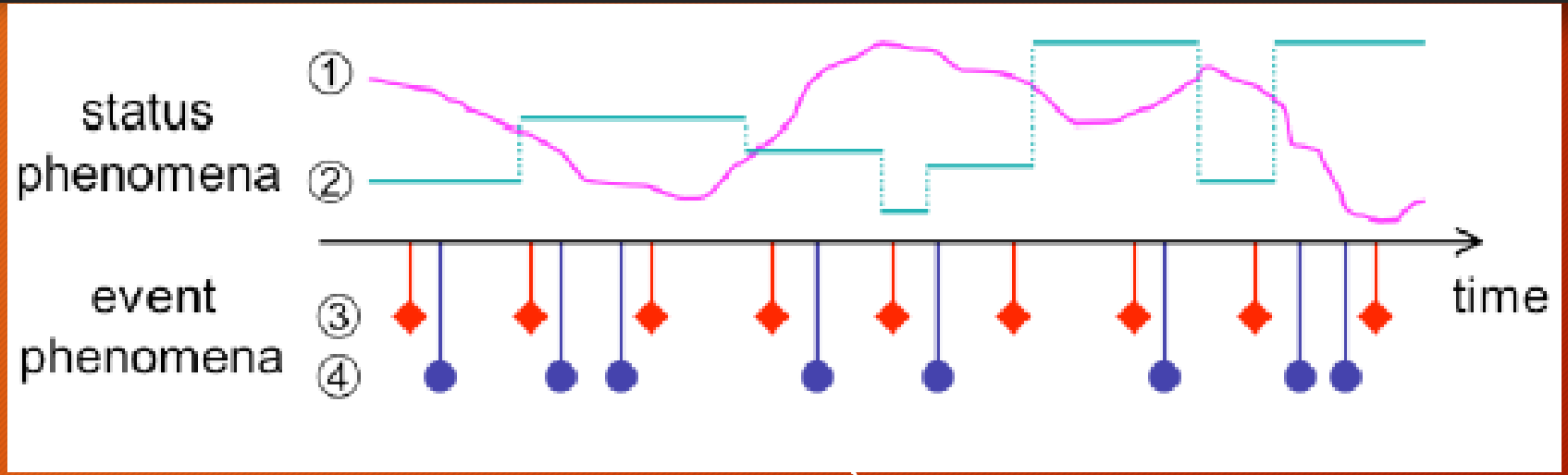
Fonctionnement

- “delegates” = callbacks
- Java: Listeners

GUI App

```
Class{
  decl data storage;

  constructor(){
      initialization code;
      create GUI controls;
      register callbacks;
  }
  main(){
      Run(new )
  }
  callback1(){
      do stuff;
  }
  callback2(){
      do stuff;
  }
…
```

Etats et événements



- 2- les variables représentent des variations par pallier
- 3- les événements peuvent avoir une origine périodique (regarder sa montre toutes les 30s)
- 4- les événements arrivent et ont un impact sur l'état

Une démarche de conception

- Une démarche de conception
- Une notation les automates
- Un processus proche de E/A (conception de bases de données)
- Un cheminement vers le code de l'application
- Pas de fossés à combler intellectuellement

Une démarche de conception

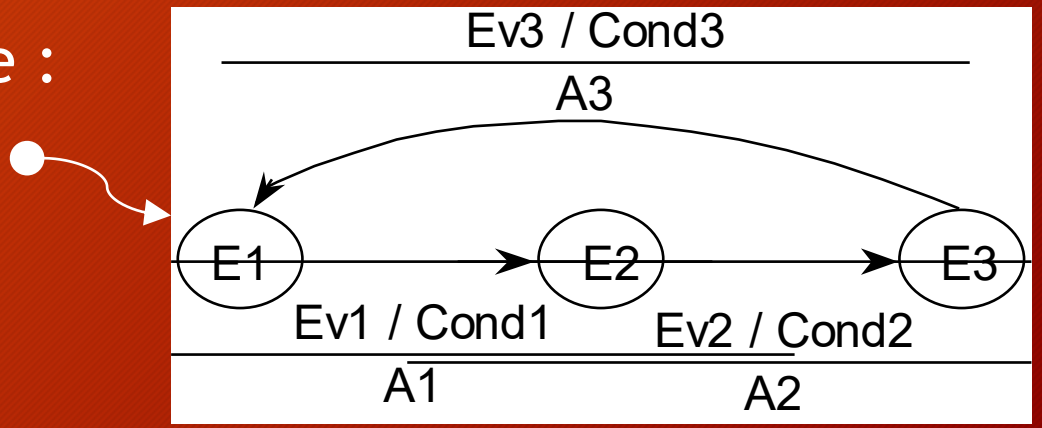
- 1) Analyse
  - a) conception de l'interface (design, choix des objets, …)
  - b) liste des événements
  - c) liste des actions
- 3) automate de comportement
- 4) Matrice états/événements
- 5) Event-handlers

Avantages

- Description complète et non ambiguë
- Analyse de propriétés
  - Comportementales
  - D'utilisabilité
- Génération de code
- Il est plus facile de prouver que de tester

Les Automates Etendus

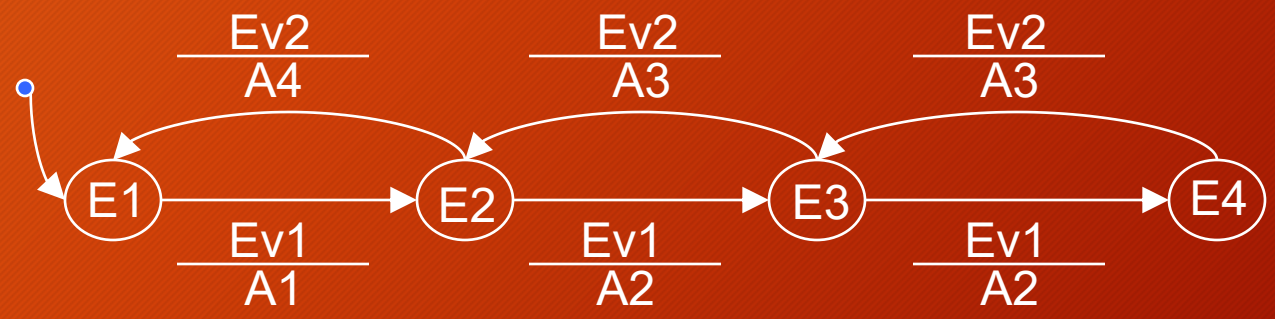
- Un automate étendu est un automate à états pouvant posséder :
  - des événements déclenchant des actions
  - des conditions de déclenchement des actions
  - des registres (variables d'états supplémentaires)
  - effectuer des actions sur les registres (affectation)
- Les événements, les conditions et les actions sont représentés
- sur les arcs sous la forme suivante :



Les Automates Etendus (2)

Exemple : un système simple

- **E = {E1, E2, E3, E4}, s0 = E1**
- **Ev = {Ev1, Ev2},**
- **A = {A1, A2, A3, A4},**
- **f : E x Ev -> A,**
  f(E1,Ev1) = A1,
  f(E2,Ev1) = f(E3,Ev1) = A2,
  f(E4,Ev2) = f(E3,Ev2) = A3,
  f(E2,Ev2) = A4
- **g : E x Ev -> E,**
  g(E1,Ev1) = g(E3,Ev2) = E2,
  g(E2,Ev1) = g(E4,Ev2) = E3,
  g(E3,Ev1) = E4,
  g(E2,Ev2) = E1.



Exemple par événement

V = {v}, v0 = 1 and v : integer

```
Handler Ev1
 Case v of
  1 : A1; v:=2; ev1 actif ev2 actif
  2 : A2; v:=3; ev1 actif ev2 actif
  3 : A2; v:=4; ev1 inactif ev2 actif
  4 : 'Interdit
 Endcase
EndHandler;
```

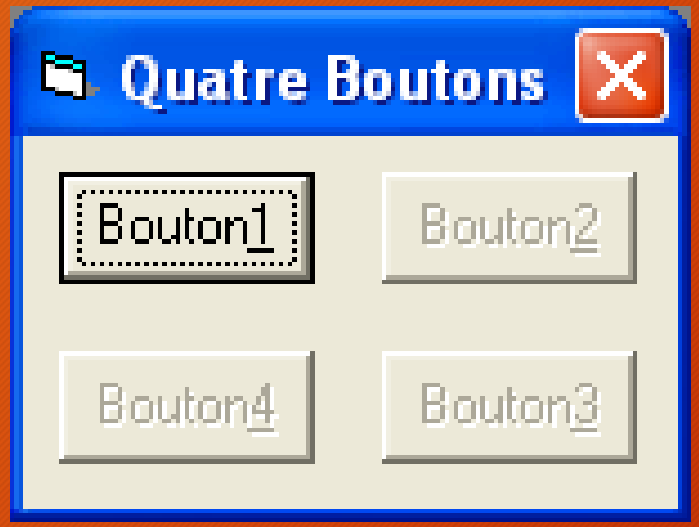
```
Handler Ev2
 Case v of
  1 : 'Interdit
  2 : A4; v:=1; ev1 actif ev2 inactif
  3 : A3; v:=2; ev1 actif ev2 actif
  4 : A3; v:=3; ev1 actif ev2 actif
 Endcase
EndHandler;
```

Tableau reccapitulatif

Protocole de communication (appli. - utilisateur)	Application = client Utilisateur = serveur	Application = serveur Utilisateur = client
Nature de l'appli.	Transformationnelle	Inter(ré)active
Contrôle	Impératif	Déclaratif
Etat du dialogue (de l'interaction)	Historique	Valeur des variables d'état

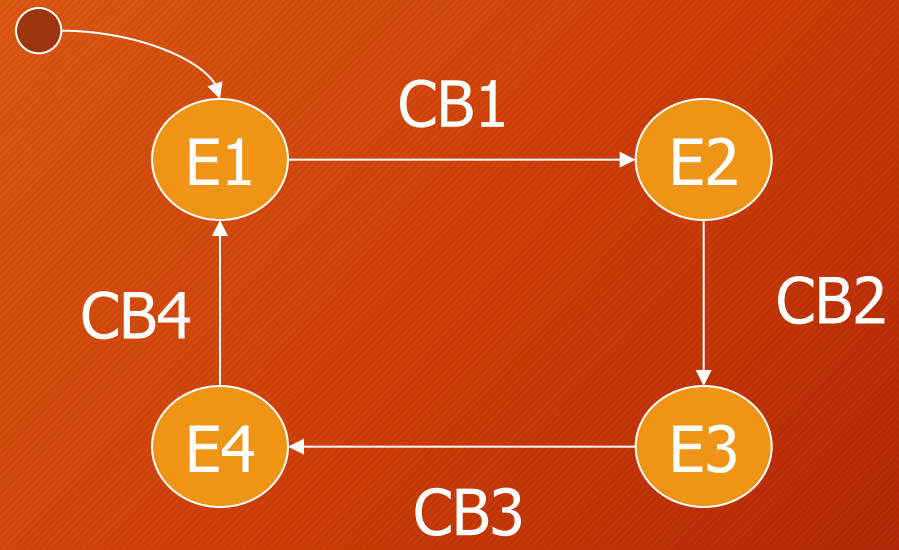
Exemple: les 4 boutons

- Spécification du comportement d'une application avec 4 boutons cycliques



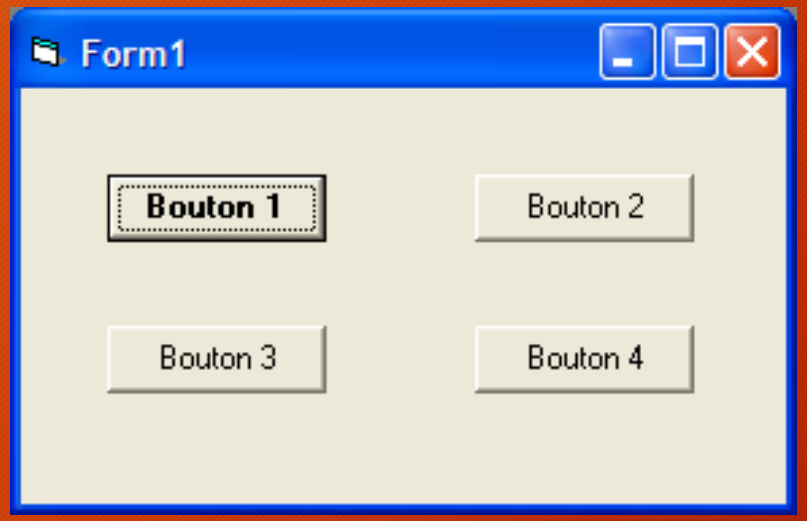
Automate Exercice 1

- 4 événements CB1, CB2, CB3, CB4



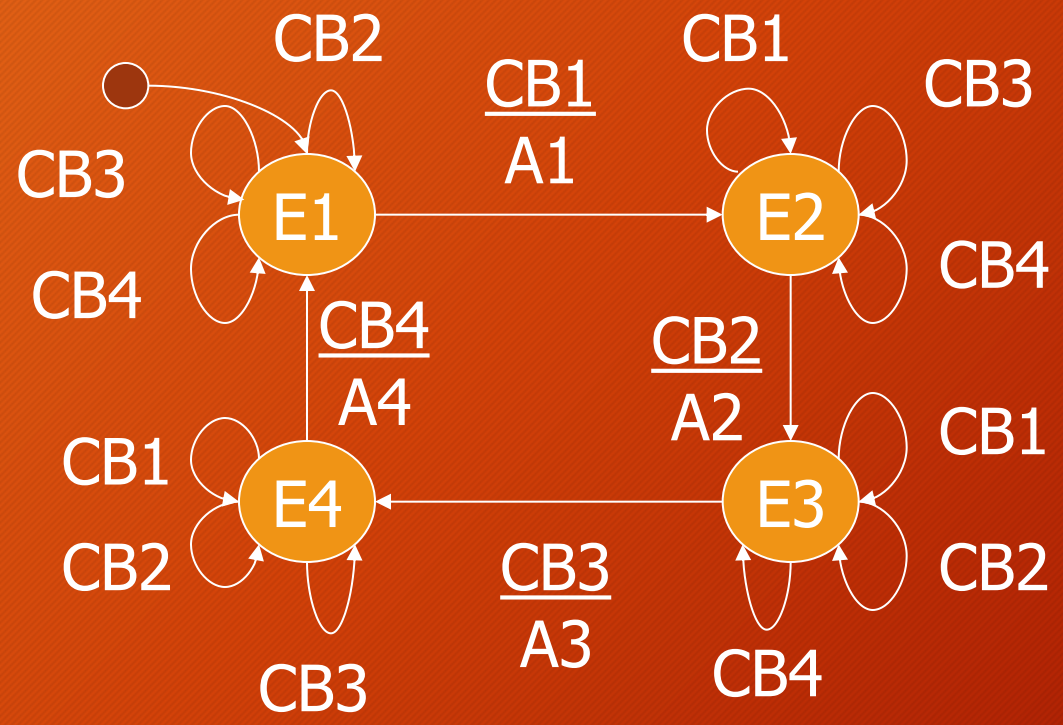
Exemple: les 4 boutons cycliques

- Spécification du comportement d'une application avec 4 boutons cycliques toujours actifs



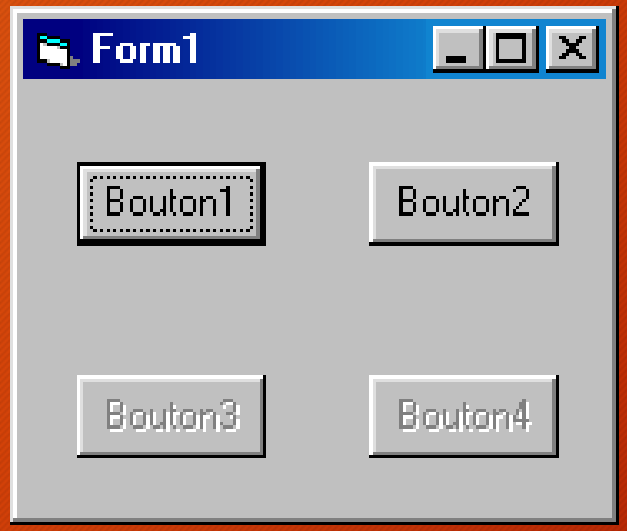
Automate Exercice 1 (2/3)

- 4 événements CB1, CB2, CB3, CB4
- 4 actions (chgt apparence boutons)



Exemple: les 4 boutons

- Spécification du comportement d'une application avec 4 boutons alternatifs



Vérification de propriétés

- P1: Au moins 2 boutons sont toujours actifs
- P2: Au plus 2 boutons sont toujours actifs
- P3: Chaque bouton peut redevenir actif à partir de n'importe quel état
  - P3.1: Quelque soit l'état il est toujours possible de trouver un chemin qui rende Bouton1 actif
  - P3.2: Quelque soit l'état il est toujours possible de trouver un chemin qui rende Bouton2 actif
  - P3.3: Quelque soit l'état il est toujours possible de trouver un chemin qui rende Bouton3 actif
  - P3.4: Quelque soit l'état il est toujours possible de trouver un chemin qui rende Bouton4 actif

Vérification de propriétés

- P4: exclusion mutuelle des boutons 2 à 2
  - P4.1: Jamais le bouton 1 et le bouton 3 ne sont actifs en même temps
  - P4.2: Jamais le bouton 1 et le bouton 4 ne sont actifs en même temps
  - P4.3: Jamais le bouton 2 et le bouton 3 ne sont actifs en même temps
  - P4.4: Jamais le bouton 2 et le bouton 4 ne sont actifs en même temps
- P5: fonctionnement par paire
  - P5.1: Si le bouton 1 est actif, alors le bouton 2 est actif
  - P5.2: Si le bouton 2 est actif, alors le bouton 1 est actif
  - P5.3: Si le bouton 3 est actif, alors le bouton 4 est actif
  - P5.1: Si le bouton 4 est actif, alors le bouton 3 est actif
- P6: initialisation
  - P.6.1: à l'initialisation les boutons 1 et 2 sont actifs
  - P.6.2: à l'initialisation les boutons 3 et 4 sont inactifs

Modélisation en logique temporelle

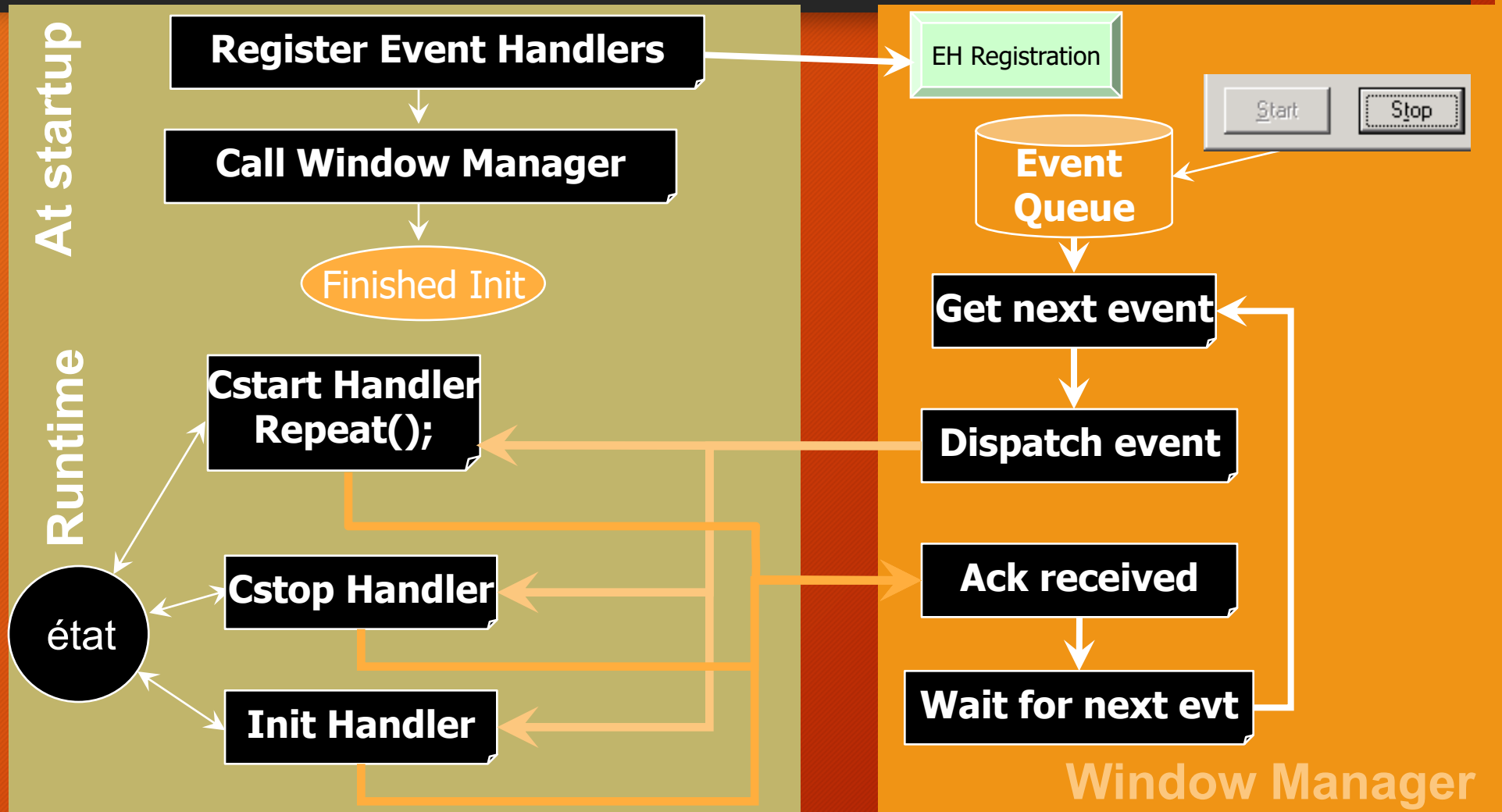
- si |= AG button1.enabled => button2.enabled
- si |= AG button2.enabled => button1.enabled
- si |= (AG button1.enabled)=False
- si |= AF button1.enabled
- si |= AF (not button3.enabled)
- si |= AG button1.enabled U button3.enabled
- si |= [AF (button1.enabled $\wedge$ button3.enabled)] = false
- si |= [EG (button1.enabled $\wedge$ button2.enabled) $\vee$ (button3.enabled $\wedge$ button4.enabled)])
- si |= [AG (button1.enabled $\wedge$ button2.enabled) $\vee$ (button3.enabled $\wedge$ button4.enabled)])

$\wedge$ (et); $\vee$ (ou); $\neg$ (non); $\Rightarrow$ (implication)

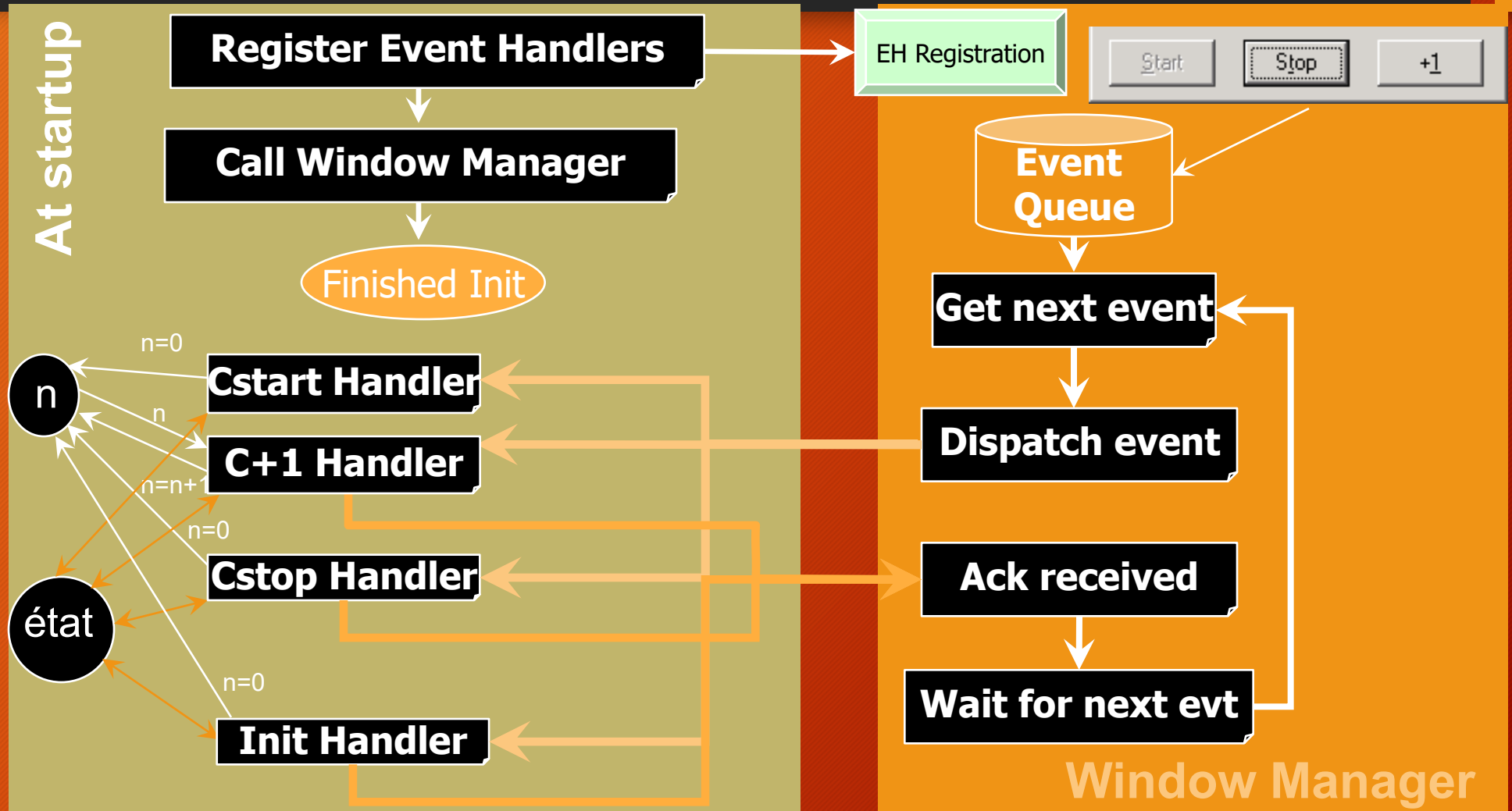
Boucle d'affichage

- 2 boutons (start, stop)
- 1 compteur
- L'utilisateur doit pouvoir interrompre à tout instant

Cas 1 (avec répétition)



Cas 2 (avec C+1 et/ou Timer)



Avance - Recule

- Possibilité d'incrémenter
- De décrémenter
- D'arrêter l'exécution
- Quand on atteint le max
  - On ne change pas la valeur affichée (max)
  - Tous les boutons sont désactivés sauf « Start »
- Quand on atteint le min
  - On ne change pas la valeur affichée (0)
  - Tous les boutons sont désactivés sauf « Start

Etat initial

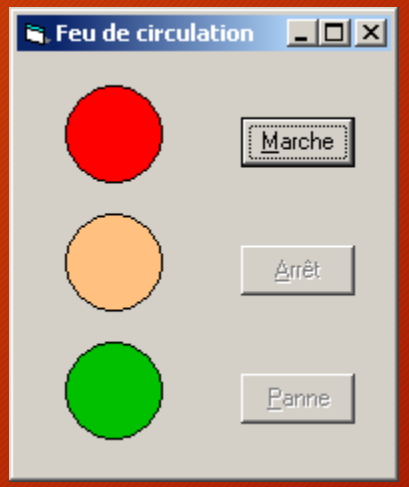
Après appui sur Start

Après appui sur marche arrière

Après appui sur stop

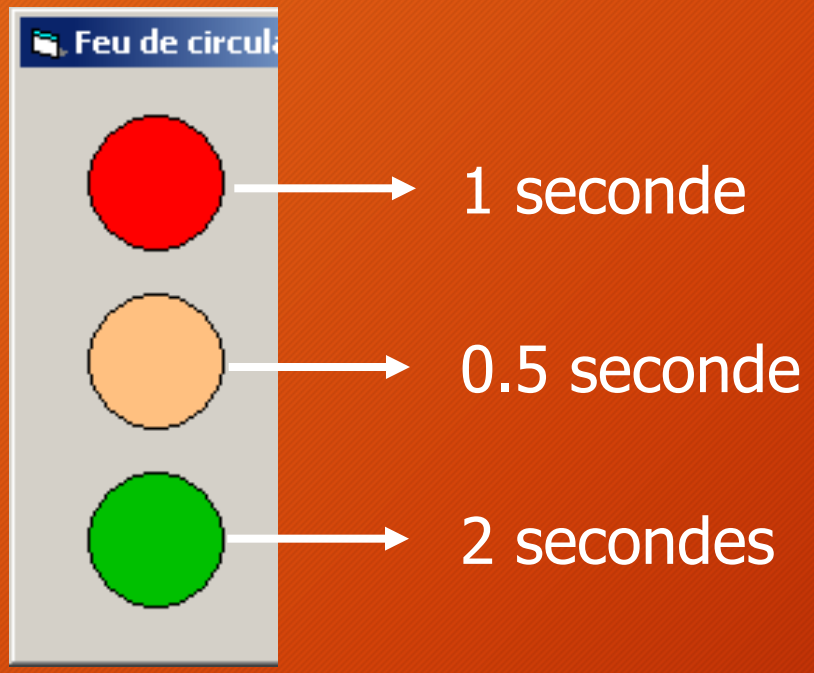
Exemple: le feu de circulation

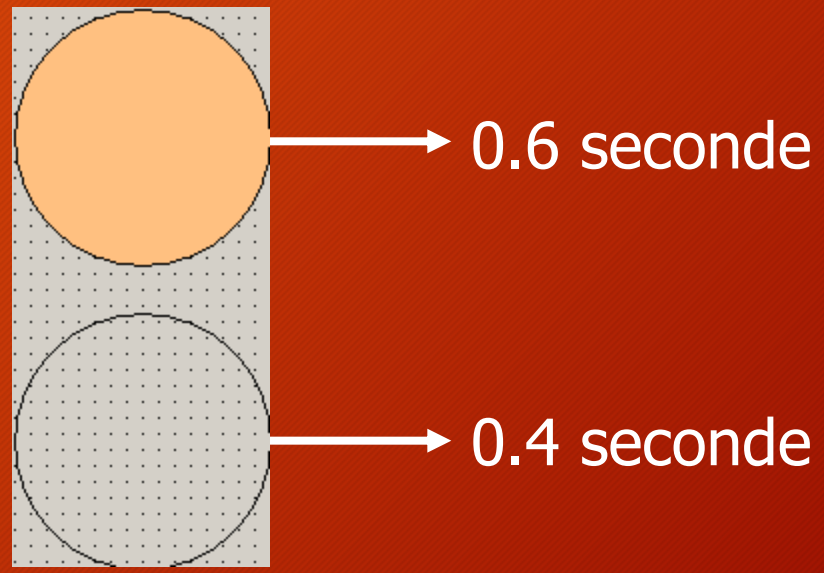
Spécification du comportement d'une application de gestion de feu de circulation



Exemple: le feu de circulation

- Aspects temporels

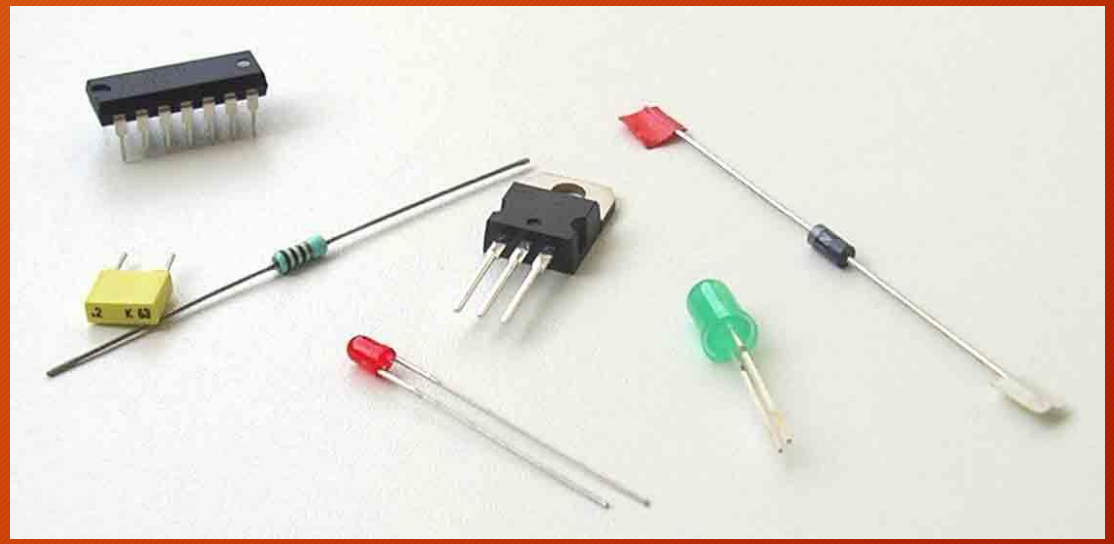




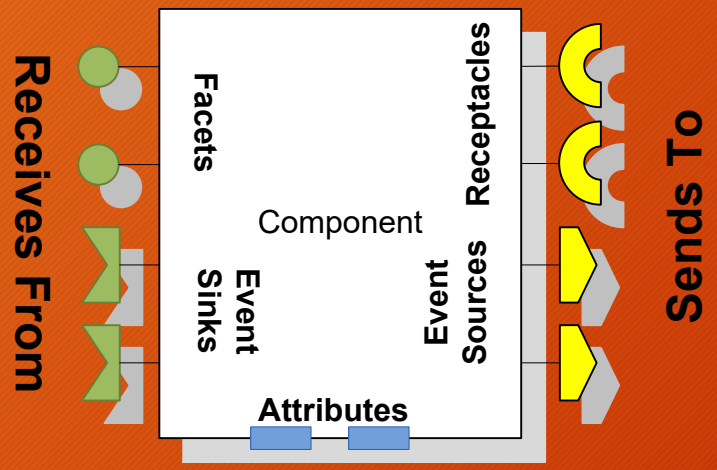
Approche par composants

Qu’est-ce qu’une bibliothèque de composants ?

- Une « boîte à outils »
- Une connectique
- Potentiellement, des moyens de paramétrisation



Exemple de Modèle : CORBA Component Model (CCM)



- Un composant est représenté comme une collection de « ports »
  - Facets : interfaces offertes
  - Receptacles : interfaces requises
  - Event Sources : événements produits
  - Event Sinks : événements consommés
  - Attributes : propriétés configurables

Exemples de composants

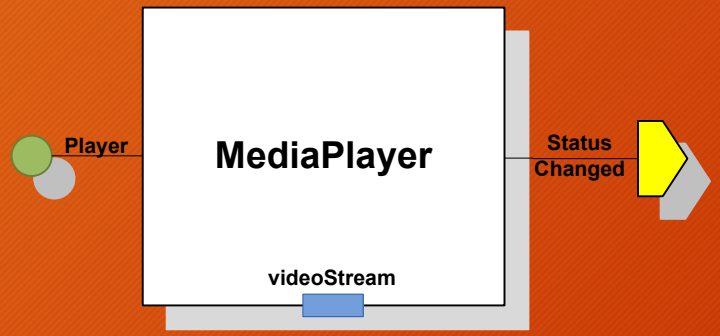
- Media Player
  - Facets

```
interface player {
    void play();
    void stop();
}
```

  - Attributes
    - videoStream
  - Event Sources
    - notification du changement d'état (playing, stopped)
- Bouton
  - Attributes
    - enabled
  - Event Sources
    - notification de son utilisation (actionPerformed)

Exemples de composants

Media Player

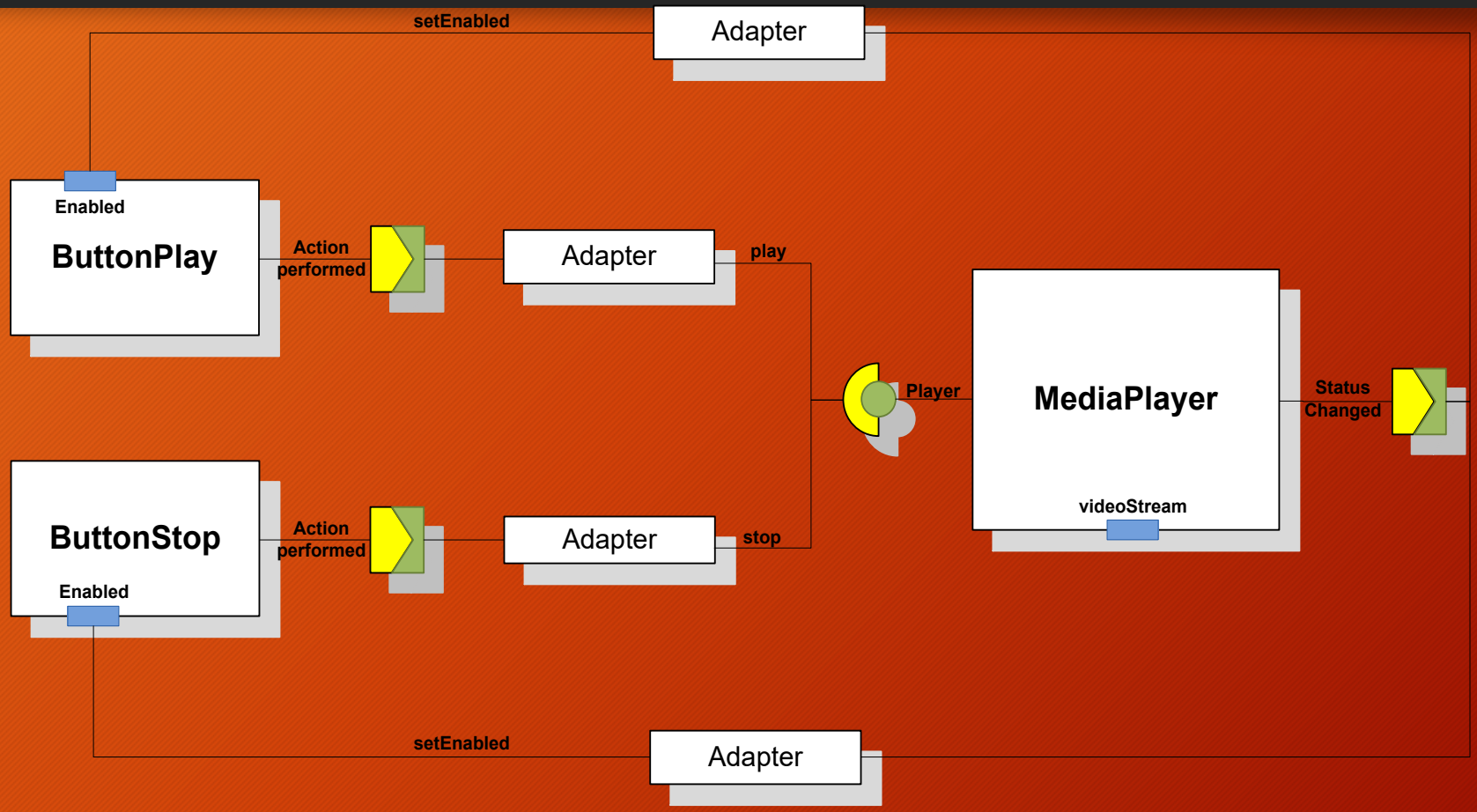


Bouton

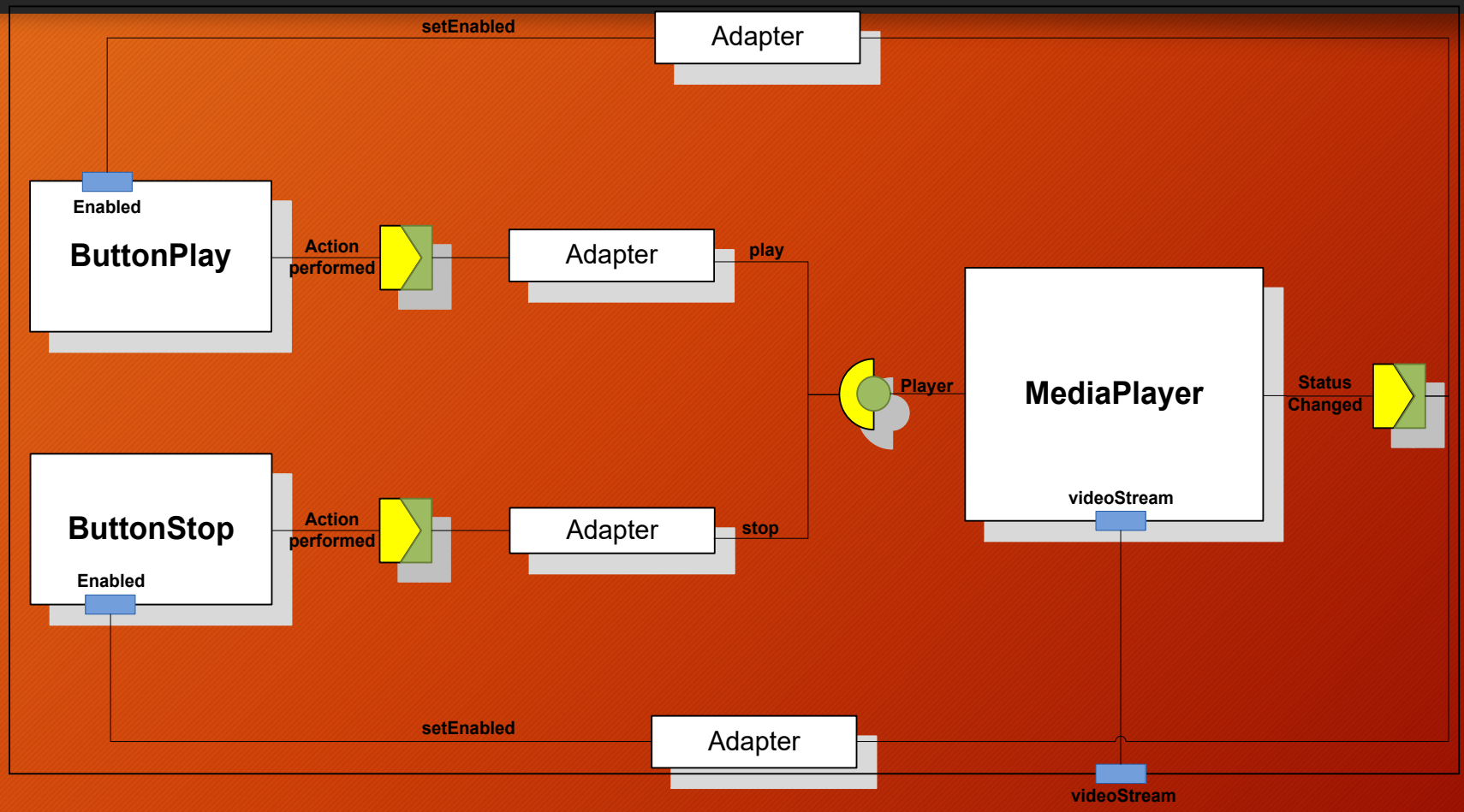
Composition : Media player standalone

- Composition du composant MediaPlayer et des boutons
- Un bouton démarre la lecture
- Un bouton stoppe la lecture

Exemple : Composition



Exemple : Nouveau composant



Impact sur le processus de développement

- Lorsqu'on commence à développer l'application, on assemble des composants
  - Encourage à commencer par le développement de l'interface
  - La gestion de la partie interaction est séparée du reste du code (modulaire ?)
- Design-time : deux métiers
  - Développement d'un composant
  - Intégration avec d'autres pour composer une application
- Utilisation d'un outil spécialisé

Conclusions sur les composants (1/2)

- Attention : la robustesse d'un système est celui de son plus faible composant
  - fiabilité des composants
  - isolement des erreurs
- Le développement de composants fiables est plus dur car l'intégration se fait ailleurs « third parties »

Conclusions sur les composants (2/2)

- La gestion des performances est cruciale
  - rapidité d'exécution
  - consommation de ressources mémoire
  - consommation de ressources physiques
  - consommation de ressources réseau

Exemples de bibliothèques de composants

Java, .net, Qt…

Critères de comparaison (1/2)

- Reprise des éléments de base du modèle de composant
  - Pour chaque critère, illustration de la technologie par des exemples
  - Code ou éléments du langage ou éléments de la bibliothèque
- Arguments pour le choix de la bibliothèque utilisée à titre d'illustration
  - On privilégie l'aspect pédagogique
  - L'important est d'être capable de se débrouiller avec n'importe quelle bibliothèque

Critères de comparaison (2/2)

- Champs d'application
  - Multiplateforme, Bureau, Web, Mobile…
- Mise en œuvre
  - Attributs
  - Event Sink
  - Event Source
  - Facet
  - Receptacle
- Aides au design time

Les bibliothèques/Langages comparés

- .net (VB, C#, …)
- Qt (C++, Python, …)
- Swing (Java)
- JavaFX (Java)

- ATTENTION : Ceci n’est pas une liste exhaustive des technologies existantes
  - Illustratif des concepts de composants
  - La majorité des mécanismes se retrouve chez les autres

Champs d’application

Champs	.net	Qt	Swing	JavaFX
Bureau	**Oui**	**Oui**	**Oui**	**Oui**
Mobile	Oui	Oui	**Oui**	Oui
Web	**Oui**	Non	Oui	Oui
« Universel »	Oui	Oui	Oui	Oui

Les attributs

.net	Qt	Swing	JavaFX
Propriétés	Propriétés	Propriétés	Propriétés
• Property est un mot du langage • Définition d'accesseur • Notifications optionnelles	• Macro Q_PROPERTY(…) • Accesseurs masqués par la macro • Notifications optionnelles	• JavaBeans • Convention de nom • Accesseurs • Mécanisme d'observation optionnel	• Compatible JavaBeans • Wrapper sur les properties incluant mécanisme d'observation et binding

Les attributs (.net)

VB.net

```
Public Property Name() As String
    get
        return name
    set
        name = value
        raiseEvent ValueChanged
End Property
```

C#.net

```
public new String Name {
 get
    {
        return name;
    }
    set
    {
        name = value;
        raiseEvent new ValueChanged()
    }
  }
}
```

Les attributs (Qt)


```
Q_OBJECT
Q_PROPERTY( string name READ getName WRITE setName NOTIFY valueChanged)

public:
    void setName(string name);
    string getName() const;
signals:
    void valueChanged(string name);
```

Les attributs (JavaBeans/Swing)


```
Private String name = «  »;

public final String getName() {
  return name;
}

public final void setName(String name) {
  String oldName = getName();
  this.name = name;
  firePropertyChanged(« name », oldName, name);
}
```

Les attributs (JavaFX)


```
private final StringProperty name = new SimpleStringProperty();

public final String getName() {
  return name.get();
}

public final void setName(String name) {
 this.name.set(value);
 fireEvent(new ValueChangedEvent(…));
}

public final StringProperty nameProperty() {
  return name ;
}
```

Event Sinks

.net	Qt	Swing	JavaFX
• Constructions du langage • handles • addHandler • XXX.<Event> += <handler>	• Call of virtual method • MyWidget::event(QEvent *event) • **OU** Utilisation de signals	• Pattern Observer • Convention de nom • Add/removeXXXListener(…) • Avec typage (EventListener, EventObject)	• Pattern observer • Typage (EventHandler, EventObject) • Template addEventHandler(T handler)

Event Sinks (.net)

Vb.net

```
Friend WithEvents Button1 As
System.Windows.Forms.Button

Private Sub Button1_Click(…) Handles Button1.Click
End Sub
```

Ou

```
Private Sub Button1_Click(…)

AddHandler Button1.Click, AddressOf Me.Button1_Click
```

C#.net

```
private void Button1_Click(object sender,
System.EventArgs e){}

Button1.Click += new
System.EventHandler(this.Button1_Click);
```

Event Sinks (Qt)

Virtual function

```
void Button1::clicked(QMouseEvent *event){}
```

Signal/slot

```
QObject::connect(
&button1, SIGNAL(clicked()),
&b, SLOT(handleEvent()));
```

Event Sinks (JavaBeans/Swing)


```
Button1.addActionListener((ActionListener)listener);
```

Event Sinks (JavaFX)


```
button1.addEventHandler((EventHandler<ActionEvent>)handler);
```

Event Sources

.net	Qt	Swing	JavaFX
• Primitives du langage • WithEvent • Event • raiseEvent • AddHandler • tout est customisable	• QCoreApplication::sendEvent() and QCoreApplication::postEvent(). • Ou Signals	• Pattern Observer • Convention de nom • Add/removeXXXListener(…) • Avec typage (EventListener, EventObject) • Gestion de la notification et des listeners manuelle	• Pattern observer • Typage (EventHandler, EventObject) • Template addEventHandler(T handler) • Peut utiliser les méthodes héritées de Node, sinon, code manuel

Event Sources (.net)

VB.net

```
Protected Overridable Sub OnTrucEvent(e As EventArgs)
    RaiseEvent TrucEvent(Me, e)
End Sub

Public Event TrucEvent As EventHandler
```

C#

```
protected virtual void OnTrucEvent (EventArgs e)
        {
            EventHandler handler = TrucEvent;
            if (handler != null)
            {
                handler(this, e);
            }
        }
public event EventHandler TrucEvent;
```

Event Sources (Qt)

Virtual function

- QMouseEvent event(QEvent::MouseButtonPress, pos, 0, 0, 0);
- QApplication::sendEvent(mainWindow, &event);

- Virtual void mousePressEvent(QMouseEvent *event)

Signal/slot

- signals:
- void trucEvent ();

Event Sources (JavaBeans/Swing)

- Private EventListenerList listeners.
- Public void addTrucListener(TrucListener listener) {
- listeners.add(Truclistener.class, listener);}

- Private void fireEvent(TrucEvent e) {
  - For(Truclistener listener : listeners.get(TrucListener.class)) {…}
- }

Event Sources (JavaFX)

- Super.addEventHandler(…)
- Super.fireEvent(…)

Facets

.net	Qt	Swing	JavaFX
Déclaration de méthodes publiques	Déclaration de méthodes publiques	Déclaration de méthodes publiques	Déclaration de méthodes publiques

Receptacles

.net	Qt	Swing	JavaFX
Appel de méthodes publiques	Appel de méthodes publiques	Appel de méthodes publiques Hors set/get/addXXXListener	Appel de méthodes publiques Hors set/get/addEventHandler

Editing Time Enhancement

.net	Qt	Swing	JavaFX
Annotations Custom Property editors	domXml()	BeanInfo Custom Property editors Annotations à partir de jdk 1.9	Rien par défaut Ustilisation de ControlsFX.PropertySheet

Customisation (.net)

VB.net

```
<DefaultEvent("ValueChanged"), DefaultProperty("Number")> _
Public Class MyControl
  Inherits Control
  ...
  <DefaultValue(False)> _
  Public Shadows ReadOnly Property TabStop() As Boolean
     get
        return name;
     set
        name = value;
  End Property

  <CategoryAttribute("Data")> _
  Public ReadOnly Property Number() As Integer
    ...
  End Property

  <Description("Raised when the Value displayed changes.")> _
  Public Event ValueChanged As EventHandler
  ...
End Class
```

C#.net

```
[DefaultEvent("ValueChanged")]
[DefaultProperty("Number")]
public class MyControl : Control {
  ...
  [DefaultValue(false)]
  public new bool TabStop {...
  get
      {
         return name;
      }
      set
      {
         name = value;
      }
  }

  [CategoryAttribute("Data")]
  public int Number {...}

  [Description("Raised when the Value displayed changes.")]
  public event EventHandler ValueChanged;
}
```

Customisation (Qt)


```
<ui language="c++"> displayname="MyWidget">
    <widget class="widgets::MyWidget" name="mywidget"/>
    <customwidgets>
        <customwidget>
            <class>widgets::MyWidget</class>
            <addpagemethod>addPage</addpagemethod>
            <propertyspecifications>
                <stringpropertyspecification name="fileName" notr="true" type="singleline"/>
                <stringpropertyspecification name="text" type="richtext"/>
                <tooltip name="text">Explanatory text to be shown in Property Editor</tooltip>
            </propertyspecifications>
        </customwidget>
    </customwidgets>
</ui>
```

Customisation (JavaBeans)


```
public class YourServletBeanInfo extends java.beans.SimpleBeanInfo {
  static java.beans.BeanDescriptor beanDescriptor = null;

  public java.beans.BeanDescriptor getBeanDescriptor() {
    if (beanDescriptor == null) {
      ParamDescriptor [] paramDescriptors = new ParamDescriptor[2];
      ParamDescriptor [] outputDescriptors = new ParamDescriptor[1];

//This parameter is set before we service the output parameter.
      outputDescriptors[0] = new ParamDescriptor("index", "loop index (0-based)",
                                    Integer.class, false, false, null);

      paramDescriptors[0] = new ParamDescriptor("numItems",
                                     "number of times to call output",
                                     Integer.class, false, false, null);
      paramDescriptors[1] = new ParamDescriptor("output",
                                      "rendered for each iteration",
                                      DynamoServlet.class,
                                      false, true, outputDescriptors);
```

La suite …

83

Quid du web ?