

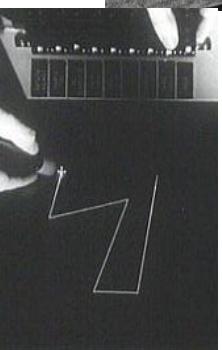
## Un peu d'histoire

3

# Développement d'applications avec IHM

Sophie Dupuy-Chessa  
Sophie.Dupuy-Chessa@univ-grenoble-alpes.fr

1963 : 1ère interface graphique SketchPad  
Interface :  
stylo optique en entrée  
+ écran en sortie  
=> Manipulation directe



## Un peu d'histoire

4

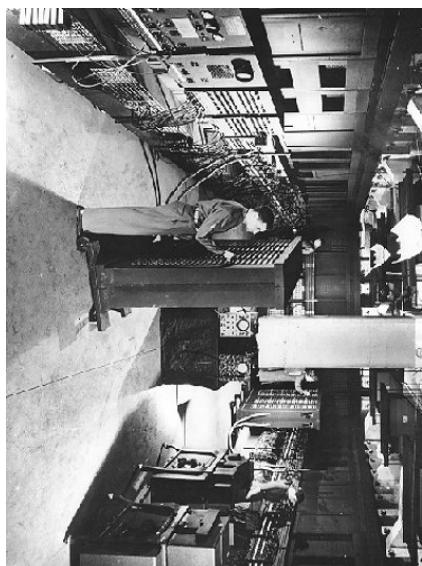
1943 : 1er ordinateur ENIAC

30 tonnes

Pièce de 23m<sup>3</sup>

0.5 M\$

Interface : branchements de fil



## Un peu d'histoire

1972 : PDPD11

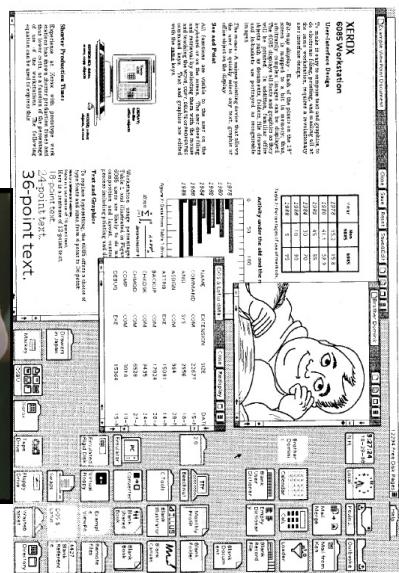
Interface en ligne de commandes  
(CLI : Command Line Interface)  
=> Shell Unix

Interface :  
Clavier en entrée  
+ imprimante à rouleau  
en sortie



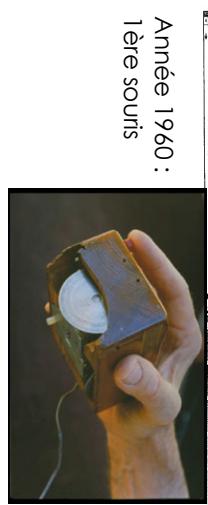
# Un peu d'histoire

Fin 1970 : Xerox Star  
Métafore du bureau



Interface :  
Clavier et souris en entrée  
+ écran en sortie

1984 : 1er Macintosh  
=> démocratisation



Année 1960 :  
1ère souris



## Introduction

**Window, Icon, Menu Pointer**

**Manipulation directe**

**Icon**

**Start button**

**Running Program**

**Task Bar**

**Language Bar**

**Free Space**

**Desktop**

**Notification Area**

**Clock**

6

## Anatomie d'une interface utilisateur

**Activer**

**Partage DVD ou CD**

**Partage d'écran**

**Partage de fichiers**

**Partage d'imprimantes**

**Partage de scanners**

**Partage web**

**Gestion à distance**

**Partage Xgrid**

**Partage Internet**

**Partage Bluetooth**

**Tout afficher**

**Nom de l'ordinateur :** Alphabet

**Modifier...**

**Rechercher dans Spotlight**

**Rechercher dans Google**

**Rechercher dans le dictionnaire**

**Rechercher dans**

**Couper**

**Copier**

**Coller**

**Orthographe et grammaire**

**Substitutions**

**Transformations**

**Parole**

**Services**

**Pour empêcher les modifications, cliquez ici.**

8

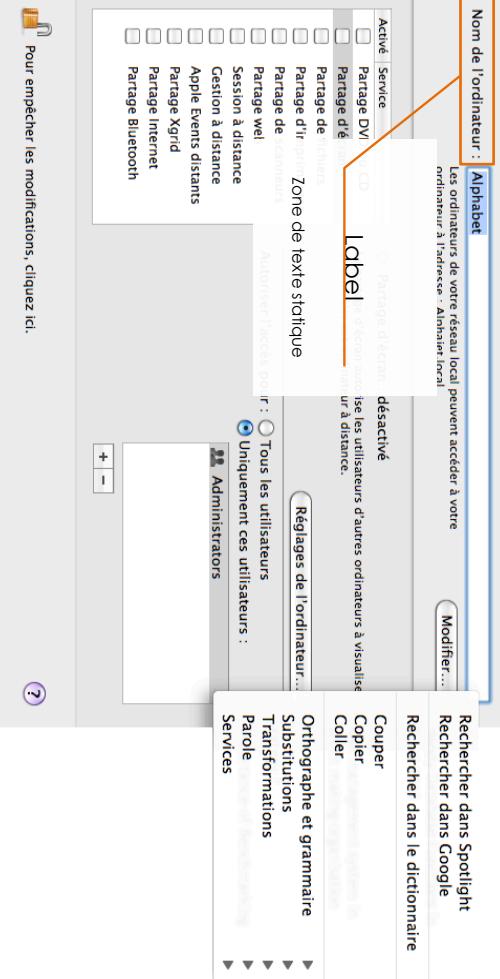
# Introduction

7



## Anatomie d'une interface utilisateur

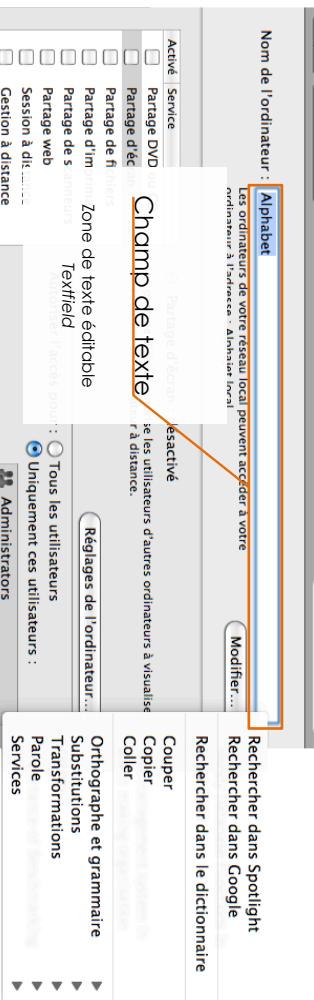
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

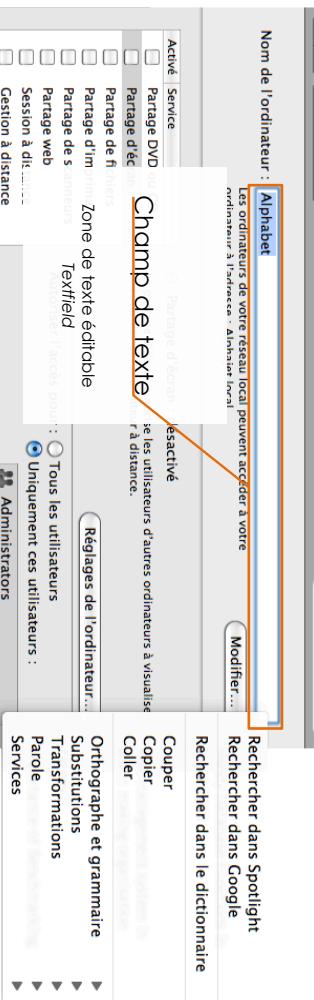
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

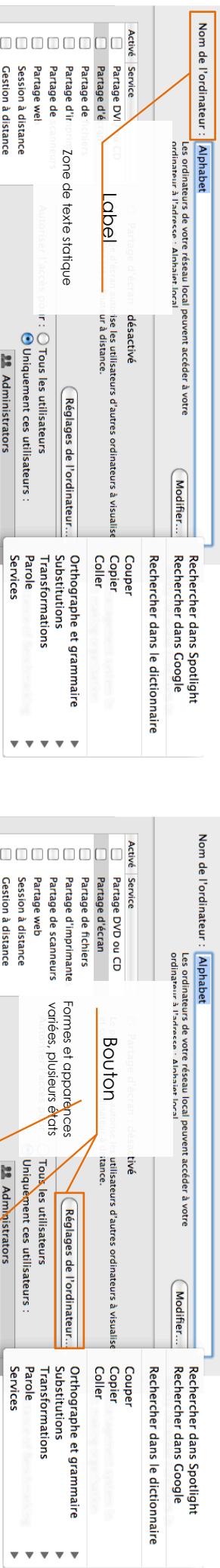
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

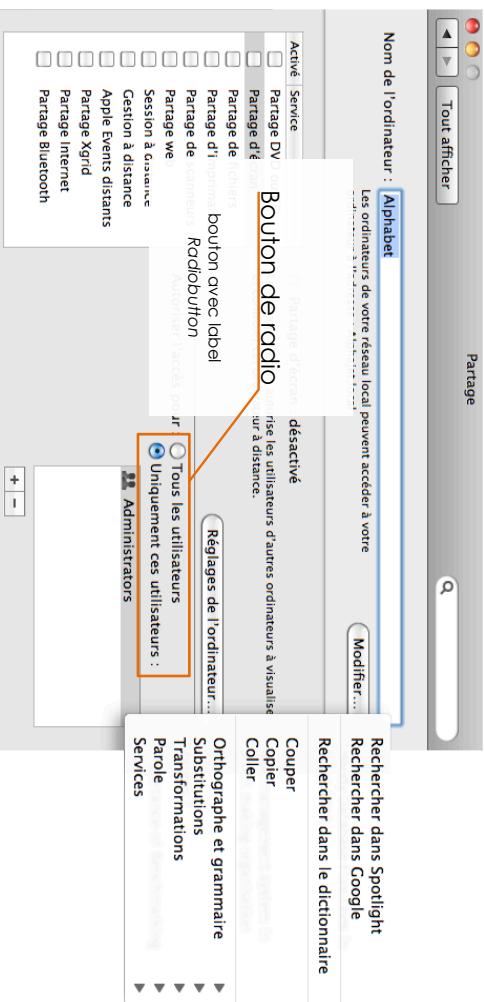
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

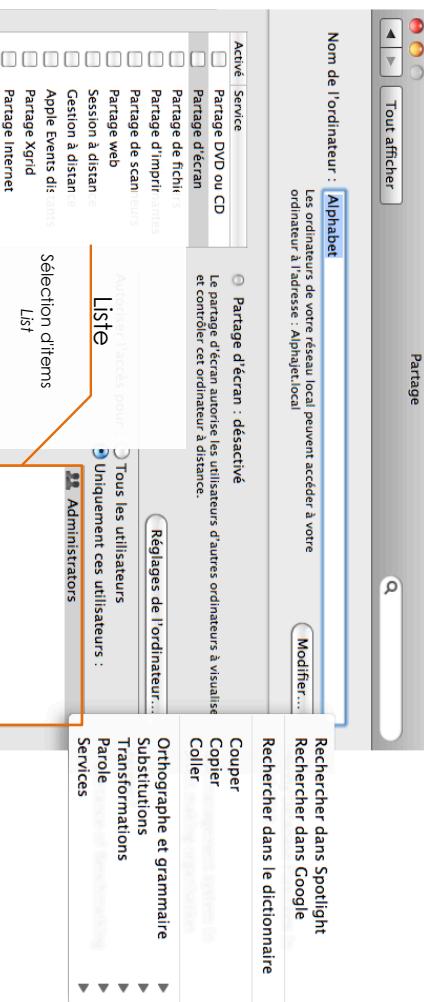
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

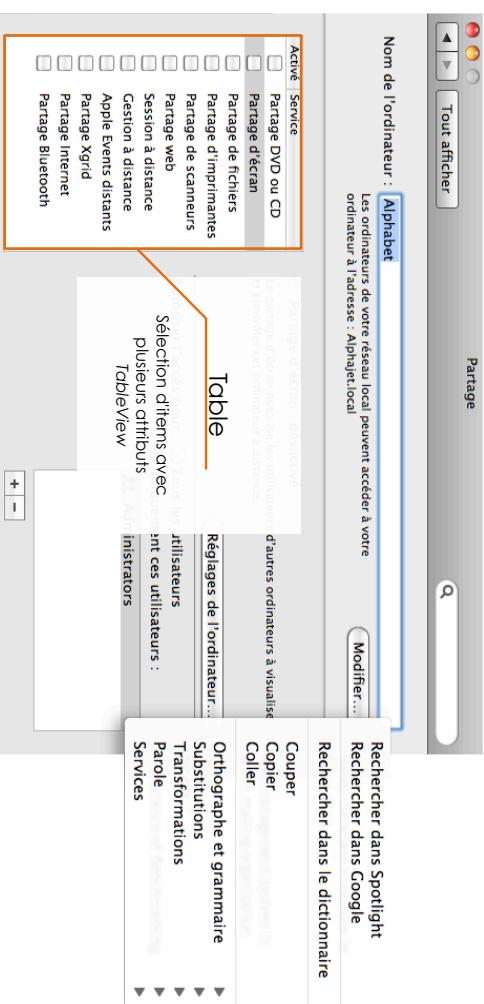
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

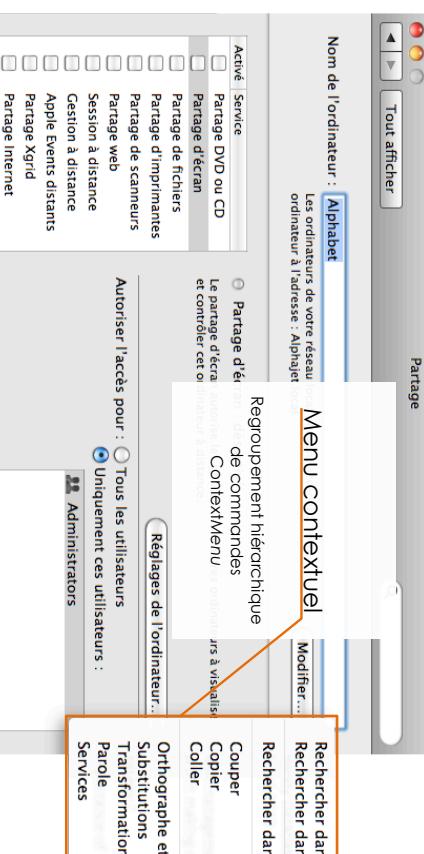
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

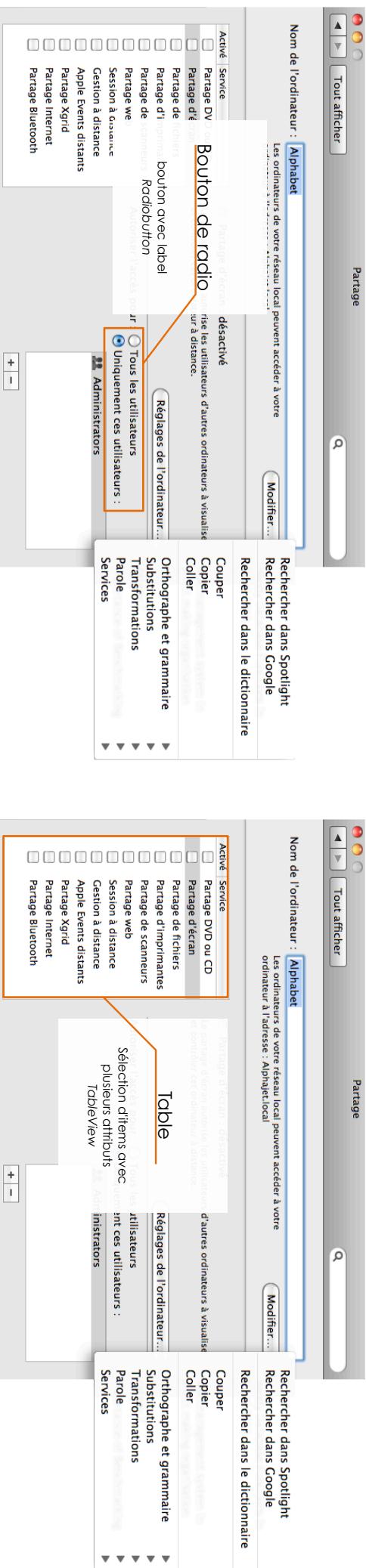
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

## Anatomie d'une interface utilisateur

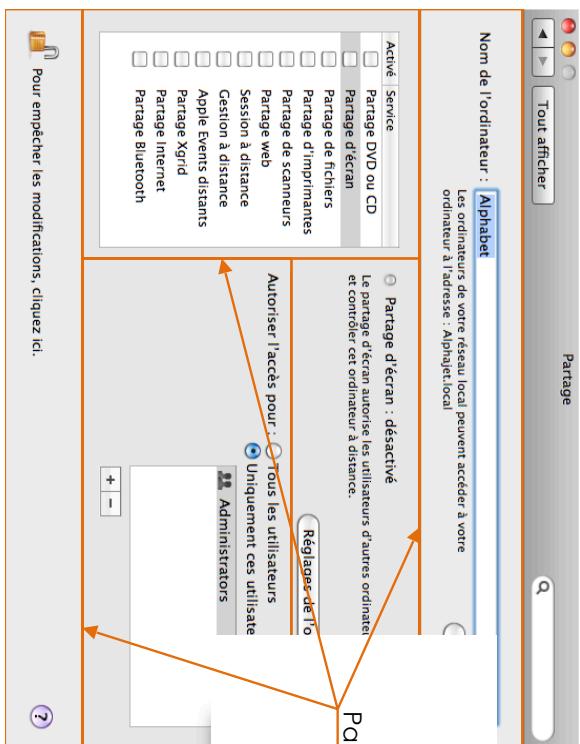
Composants d'interface graphique (widget ou control)



Pour empêcher les modifications, cliquez ici.

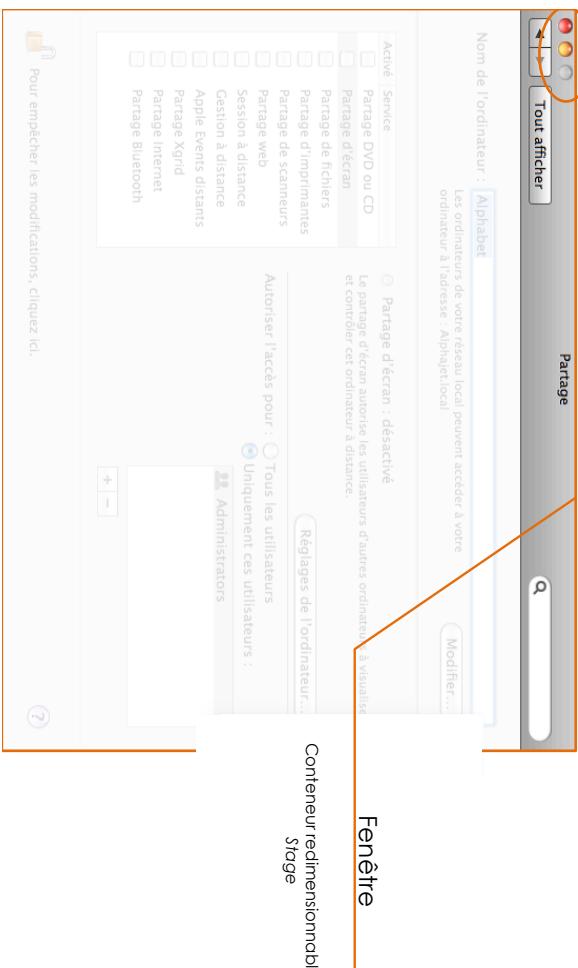
## Anatomie d'une interface utilisateur

Conteneurs (containers) : regroupement de composants



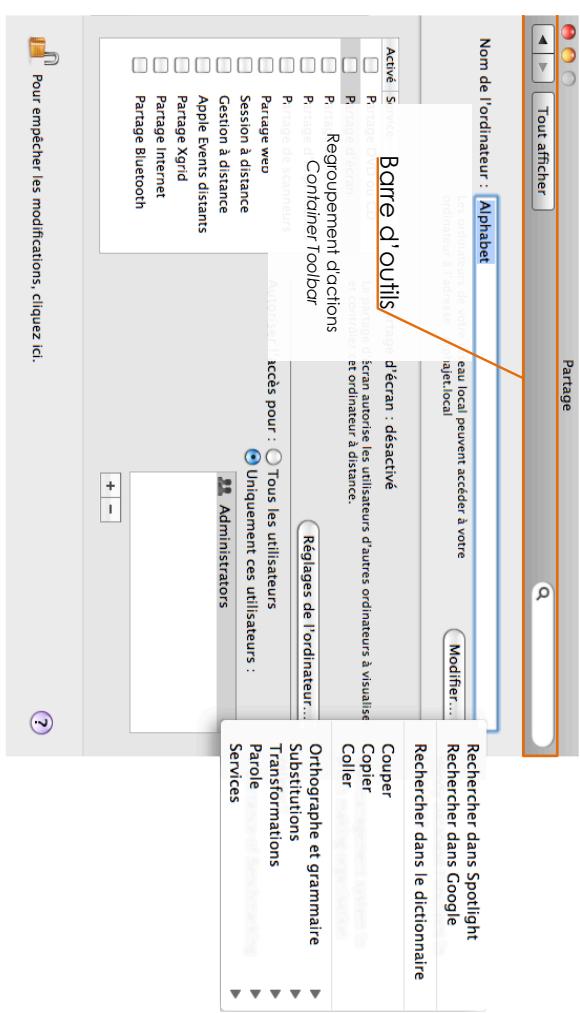
## Anatomie d'une interface utilisateur

Conteneurs (containers) : regroupement de composants



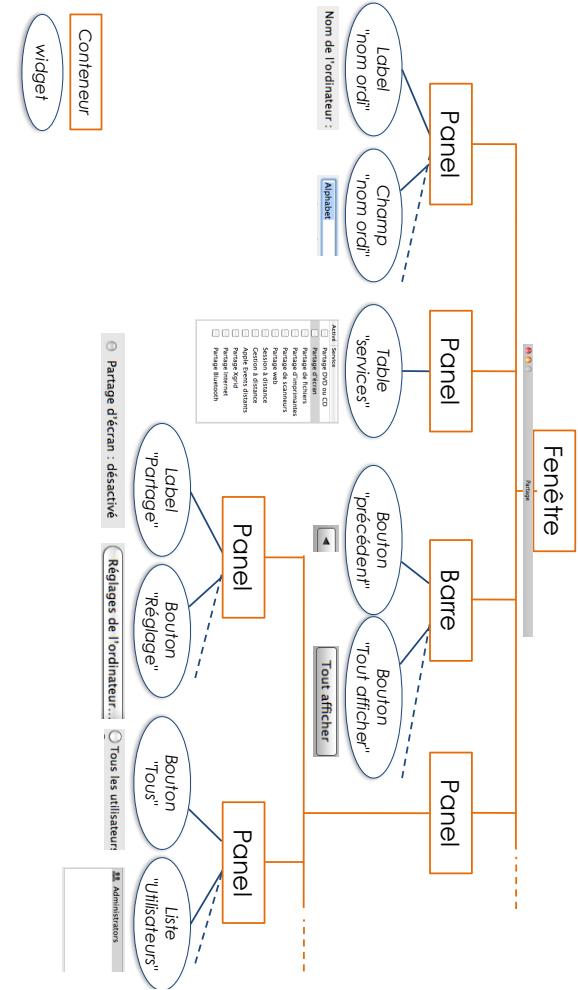
## Anatomie d'une interface utilisateur

Conteneurs (containers) : regroupement de composants



## Anatomie d'une interface utilisateur

Organisation hiérarchique



## Bibliothèques logicielles GUI

Natives, mono-plateforme

- Windows
- Cocoa (OS X, iOS)
- X11/Motif (Unix)



## JavaFX - Création d'une application

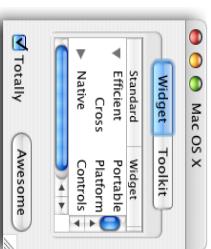
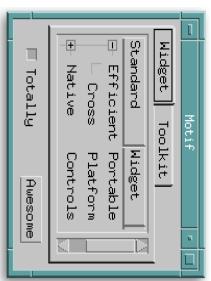
3 classes de base

### • Application : la classe principale de l'application

- Hérite de la classe javafx.application.Application
- Gère tout le cycle de vie de l'application (ouverture des fenêtres, initialisations, le démarrage et la fin de l'application, etc)

### Multi-plateforme

- JavaFX
- Java AWT/Swing
- wxWidgets
- Tk
- Qt
- GTK+



### • Stage : la fenêtre principale de l'application

- Scene : l'interface qu'il faut associer à la fenêtre

#### Node

## JavaFX

JavaFX : bibliothèque graphique qui permet de créer des interfaces utilisateur pour Java

<https://docs.oracle.com/javase/8/javafx/api/toc.htm>

22

## JavaFX - Métaphore du théâtre

Les éléments structurels principaux d'une application JavaFX s'appuient sur la métaphore d'une salle de théâtre

- Stage : l'endroit où a lieu l'action, où se déroule la scène

► la fenêtre principale de l'application

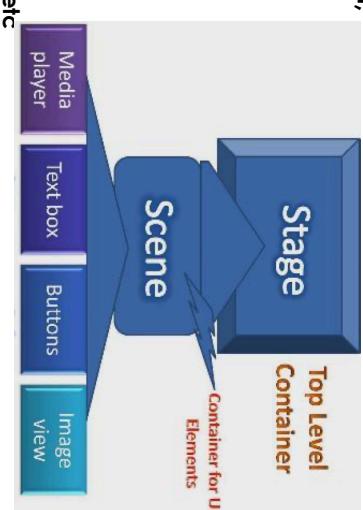
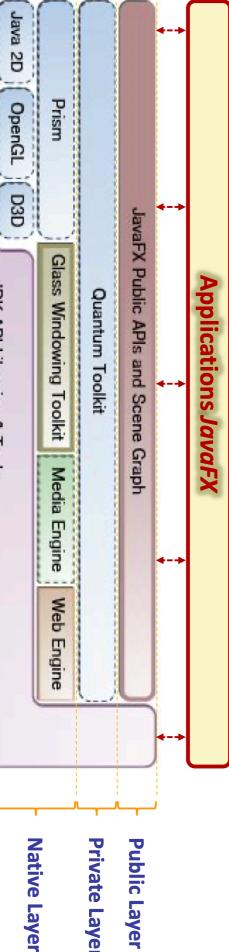
- Scene : Tableau ou séquence faisant intervenir les acteurs

► l'interface qu'il faut associer à la fenêtre

- Node (container, controls, ..) : acteurs, figurants, éléments du décors, lumière etc qui font partie de la scène en train d'être jouée

24

## Architecture technique

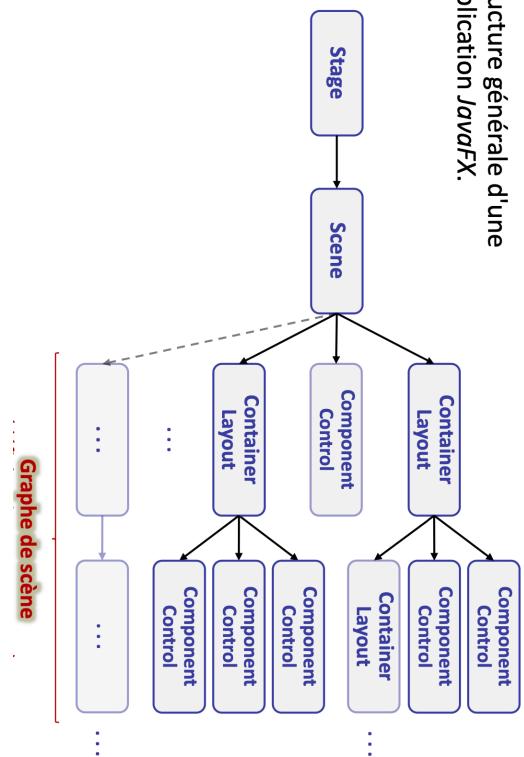


23

## Graph de scène

Un graphe de scène décrit les éléments de la scène et leur structuration

## ■ Structure générale d'une application /src/FY



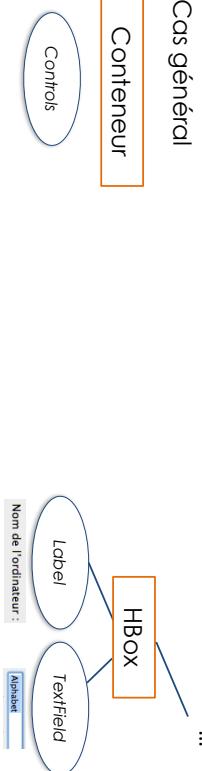
26

## Graph de scène

Un graphe de scène est un graphe acyclique orienté (arbre orienté) avec :

- une racine qui correspond à un conteneur
  - des noeuds (Node) qui sont des éléments de l'interface (conteneurs ou composants graphiques de l'interface)
  - des arcs qui représentent les relations parent-enfant

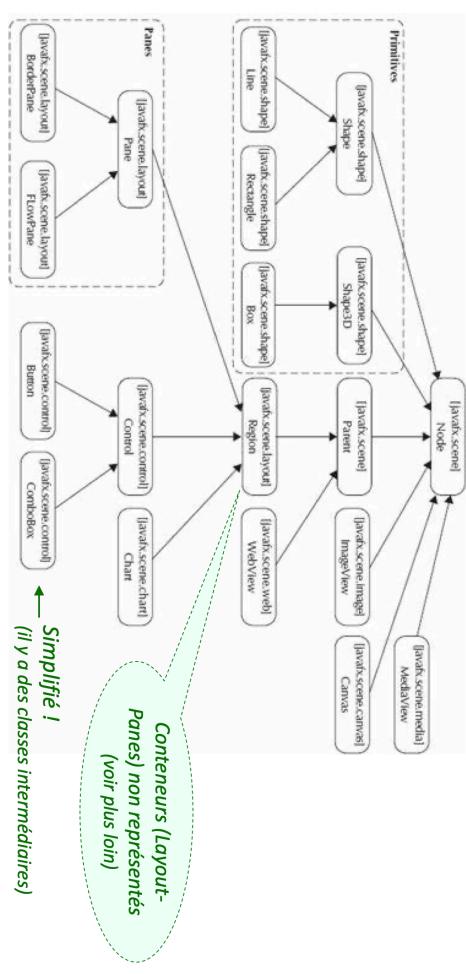
Cas général



25

## Graph de scène

Les éléments du graphe de scène sont des Node :



## Différents types de Node

### 3 types de Node :

- Formes primitives (Shape) 2D ou 3D

- **Conteneurs** (Layout-Pane) qui se chargent de la disposition (layout) des composants enfants et qui ont comme classe parente Pane.

## • Composants

- Composants standard (**Controls**)
  - Composants spécialisés qui sont :

domaine particulier

► lecteur multimédia, navigateur web, ...

28

## Différents types de Node - Formes primitives

29

## Différents types de Node - Composants

## • Composants standard (Controls)

- **Formes primitives** (Shape)

2D OU 3D

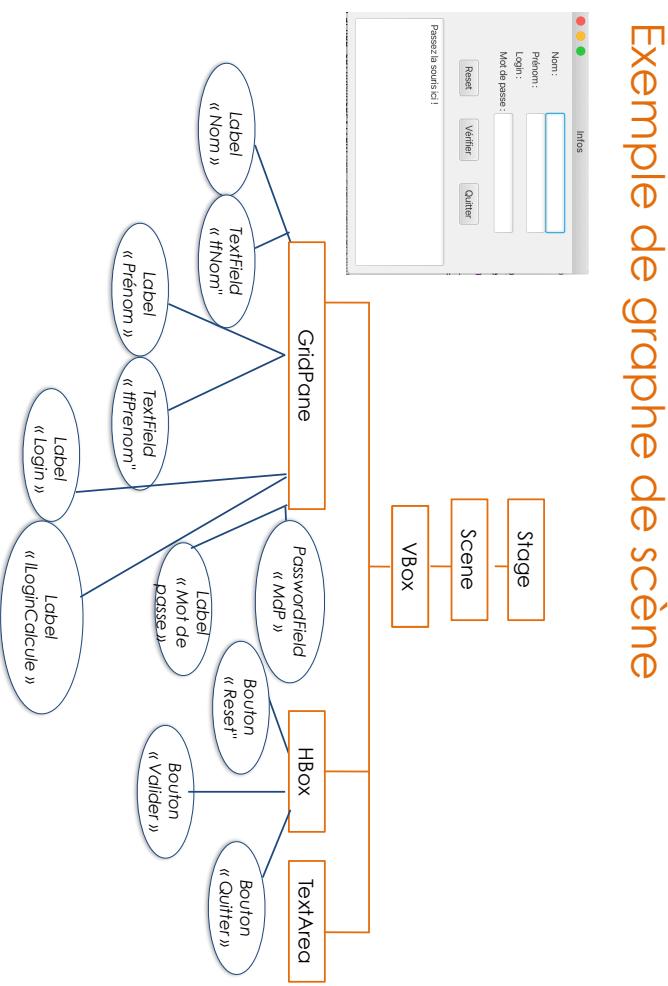
- ## ► Line, Circle, Rectangle, Box, ...



[https://docs.oracle.com/javafx/2/ui\\_controls/overview.htm](https://docs.oracle.com/javafx/2/ui_controls/overview.htm)

particulier

- MediaView, WebView, ImageView, Canvas, Chart, ...

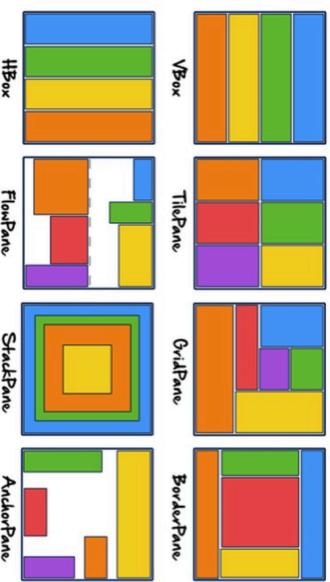


## Différents types de Node - Conteneurs

୪

## ► Containers

- AnchorPane, GridPane, VBox, ...



[https://docs.oracle.com/javafx/2/layout/builtin\\_layouts.htm](https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm)

## Différents types de Node - Formes primitives

31

# Création d'une scène

33

## Démo IntelliJ - Scene Builder

- 3 manières de créer une interface (cf TP)
  - Code JavaFX pur
  - Code JavaFX + fichier XML pour l'interface
  - Code JavaFX + fichier XML pour la structure de l'interface + fichier CSS pour des éléments visuels (couleurs,...)



34

## Rendre la scène active

Pour l'instant, il n'y a qu'un visuel. Il manque quoi faire quand l'utilisateur déclenche un événement.

Un événement est **une action de l'utilisateur**

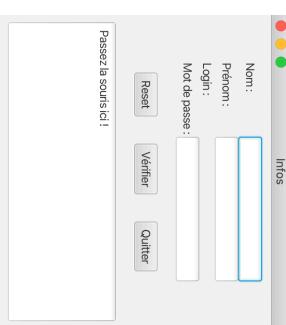
- ▶ déplacement de la souris, clic sur un bouton, appui sur une touche du clavier, ...



Action de l'utilisateur via un dispositif



Réception d'un événement type "clic de souris"



35

# Événement en JavaFX

35

- ▶ En JavaFX :
  - Un événement correspond à une **structure de données** (la classe javafx.event.Event) qui contient la source de l'événement, la cible et le type de l'événement
  - **La source est l'élément sur lequel l'événement s'est produit et qui l'envoie**

- Souris, touche, ...

- **La cible est le noeud sur lequel l'événement s'est produit**

- Bouton, case à cocher,...

- **Le type d'événement correspond à une classification**

- Clic souris, touche pressé, déplacement souris (drag),...

=> C'est JavaFX qui reçoit les événements et les transmet grâce à un gestionnaire d'événements (EventHandler).

## Événements : réagir aux actions

© Yann Laurillau

# Programmation événementielle

Action de l'utilisateur via un dispositif

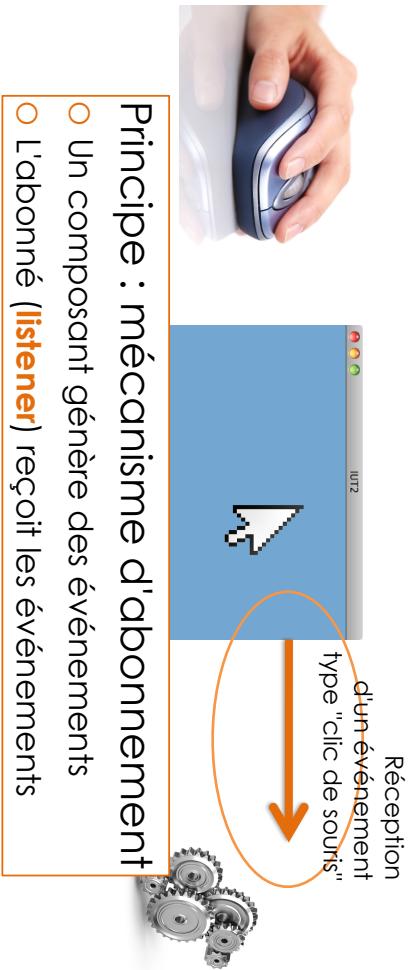


Réception d'un événement type "clic de souris" sur les événements.



© Yann Laurillau

## Événements : réagir aux actions



38

## Programmation événementielle

- On prépare le code à exécuter en l'associant à l'événement que l'on souhaite traiter
- On attend que le processus de surveillance des événements (Event Thread) détecte un événement à traiter

```

EventProg {
    Initialize();
    CreateUI();
    RegisterCallback();
    StartEventLoop(); // Thread
}

callback1() // Button clicked
code1;
callback2() // Key pressed
code2;
callback3() // Window resized
code3
...
// Pinch gesture
}

```

Pseudo-code correspondant aux principes de la prog. Événementielle en prog. impérative

Pseudo-code de programmation d'une interface

40

### Principe : mécanisme d'abonnement

- Un composant génère des événements
- L'abonné (**listener**) reçoit les événements

# Événements : réagir aux actions

# Exemple - Code Java Pur

```

    ...
    TextField tfNom = new TextField();
    Label labelLoginCalcule = new Label();
    PasswordField pmdp = new PasswordField();

    ...
    Button btnReset = new Button("Reset");
    ...
    @Override
    public void handle(ActionEvent event) {
        tfNom.setText("");
        tfPrenom.setText("");
        labelLoginCalcule.setText("");
        pmdp.setText("");
    }
}

```



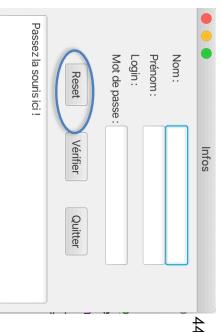
## Exemple - Code Java + fxml

### ► Fichier fxml

```

<VBox xmlns="http://javafx.com/fxml/1"
      xmlns:fx="http://javafx.com/fxml/shared"
      fx:controller="com.example.demo.TP2_Exo2Controller">
    ...
    <children>
        <Button alignment="CENTER" fillHeight="false" prefHeight="50.0" prefWidth="200.0" spacing="30.0" />
        </children>
    ...
</VBox>

```



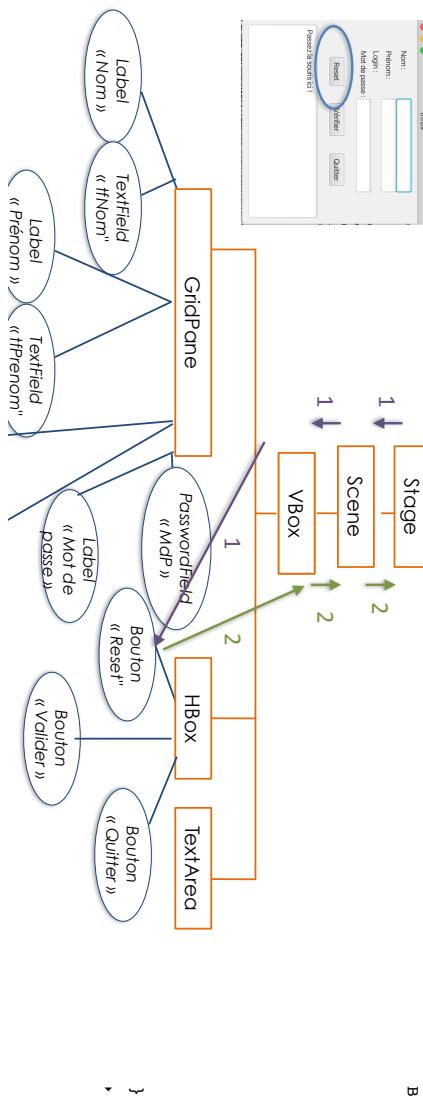
44

# Propagation d'un événement

- 1.L'événement se propage d'abord vers le bas, depuis le noeud Racine Stage jusqu'à la cible.

- Il peut être éventuellement filtré.

- 2.Il remonte ensuite de la cible vers le Stage. Les gestionnaires d'événements enregistrés sont exécutés dans l'ordre de passage.



42

## Gestion des événements

Pour gérer un événement, il faut créer un **récepteur d'événements** (**Event Listener**), et l'enregistrer sur les nœuds du graphe de scène où l'on souhaite intercepter l'événement et effectuer un traitement.

- Récepteur d'événements pour nous : gestionnaire d'événements

On dans le fichier fxml : **fx:controller**

**Ex :** `fx:controller="com.example.demo.TP2_Exo2Controller"`

- Enregistrement d'un récepteur d'événement

**Composant.actionsSurEvenement(EventHandler)**

**Ex :** `bunreset.setOnAction(new EventHandler<ActionEvent>() { tfPrenom.setOnKeyTyped(new EventHandler<KeyEvent>() { tacachee.setOnMouseEntered(new EventHandler<MouseEvent>() {`

Ou avec **SceneBuilder** / fichier fxml

**Ex :** `<Button mnemonicParsing="false" onAction="#resetButtonAction" text="Reset" />`

- Méthode de traitement de l'événement

**Handler :** `public void handle(ActionEvent event)`

Ou méthode déclarée dans **SceneBuilder**

**Ex :** `private void resetButtonAction(ActionEvent event) {`