Les événements

LES ÉVÉNEMENTS LOCAUX

Composants et événements

- Un composant peut :
  - Posséder des méthodes publiques permettant de changer son contenu
  - Posséder des méthodes publiques permettant de récupérer son contenu
  - Emettre des événements personnalisés que son parent peut notamment écouter : ils sont qualifiés d'événements « locaux »
- Plusieurs techniques de dispatch d'événement
  - Par dispatch local dans la Scene : « classique »
  - Par appel d'un EventHandler personnalisé : « databinding »

GESTION ÉVÉNEMENTIELLE

Classes, types, et workflow

- Les événements sont basés sur la classe Event
  - Sous-classes plus spécifiques : MouseEvent, ScrollEvent, etc...
- Chaque événement a un type (EventType)
  - Les types peuvent être hierarchisés
  - Exemple : InputEvent.ANY > MouseEvent.ANY > MouseEvent.MOUSE_RELEASED
- La diffusion d'événement se fait en 4 phases
  - Sélection de la cible
  - Détermination du chemin
  - Phase de capture
  - Phase de remontée (*event bubbling)*

Gestion événementielle

Description des phases

- Sélection de la cible
  - On détermine sur quel nœud l'interaction a eu lieu (le plus précisément et spécifiquement possible, les nœuds en avant-plan ayant la priorité).
  - Cette cible doit implémenter EventTarget, mais c'est déjà le cas pour les Window, Scene et Node.
- Détermination du chemin
  - L'EventTarget est chargé de définir par quel chemin l'événement va être diffusé.
  - L'implémentation par défaut sur Node renvoie un chemin allant de la Stage au nœud lui-même.

Gestion événementielle

Description des phases

- L'événement parcours le chemin vers la Node cible (phase de capture), puis remonte vers son point d'origine (phase de remontée)
- En descendant, il est traité par d'éventuels Filters
- En remontant, il est traité par d'éventuels Handlers
- Ceux-ci peuvent choisir de consommer l'événement (event.consume())
  - Il n'ira pas plus loin dans l'arborescence
  - Il entamera une remontée immédiatement dans le cas d'un Filter

GESTION ÉVÉNEMENTIELLE

Workflow

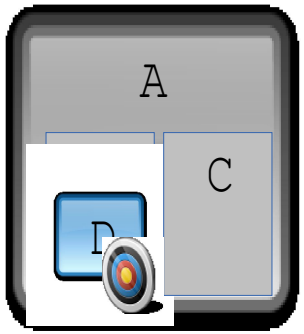
Evénement
(ex: appui sur une touche)

GESTION ÉVÉNEMENTIELLE

Workflow

Evénement
(ex: appui sur une touche)

1. Sélection



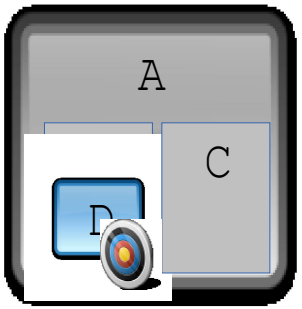
GESTION ÉVÉNEMENTIELLE

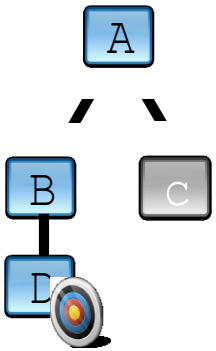
Workflow

Evénement
(ex: appui sur une touche)

1. Sélection

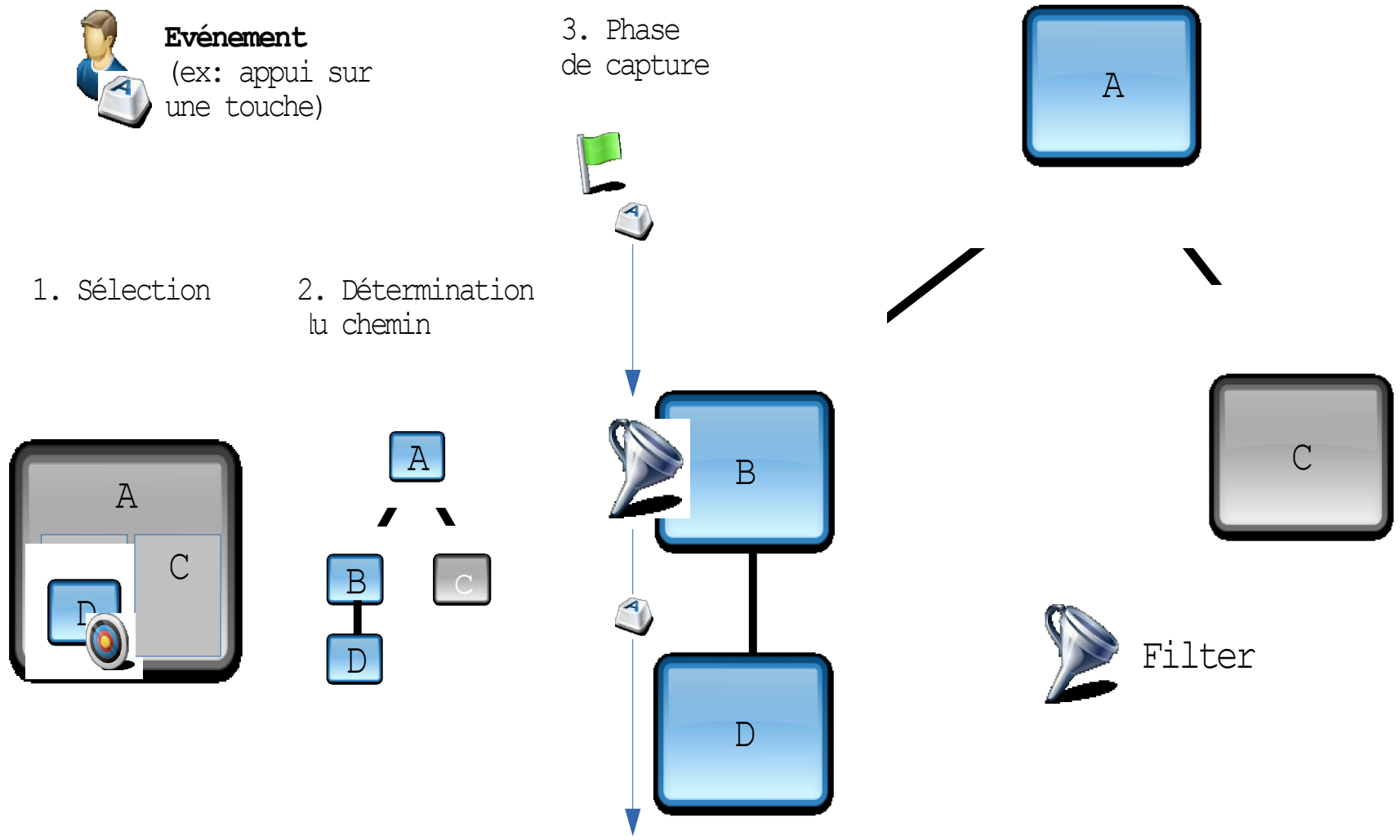
2. Détermination du chemin





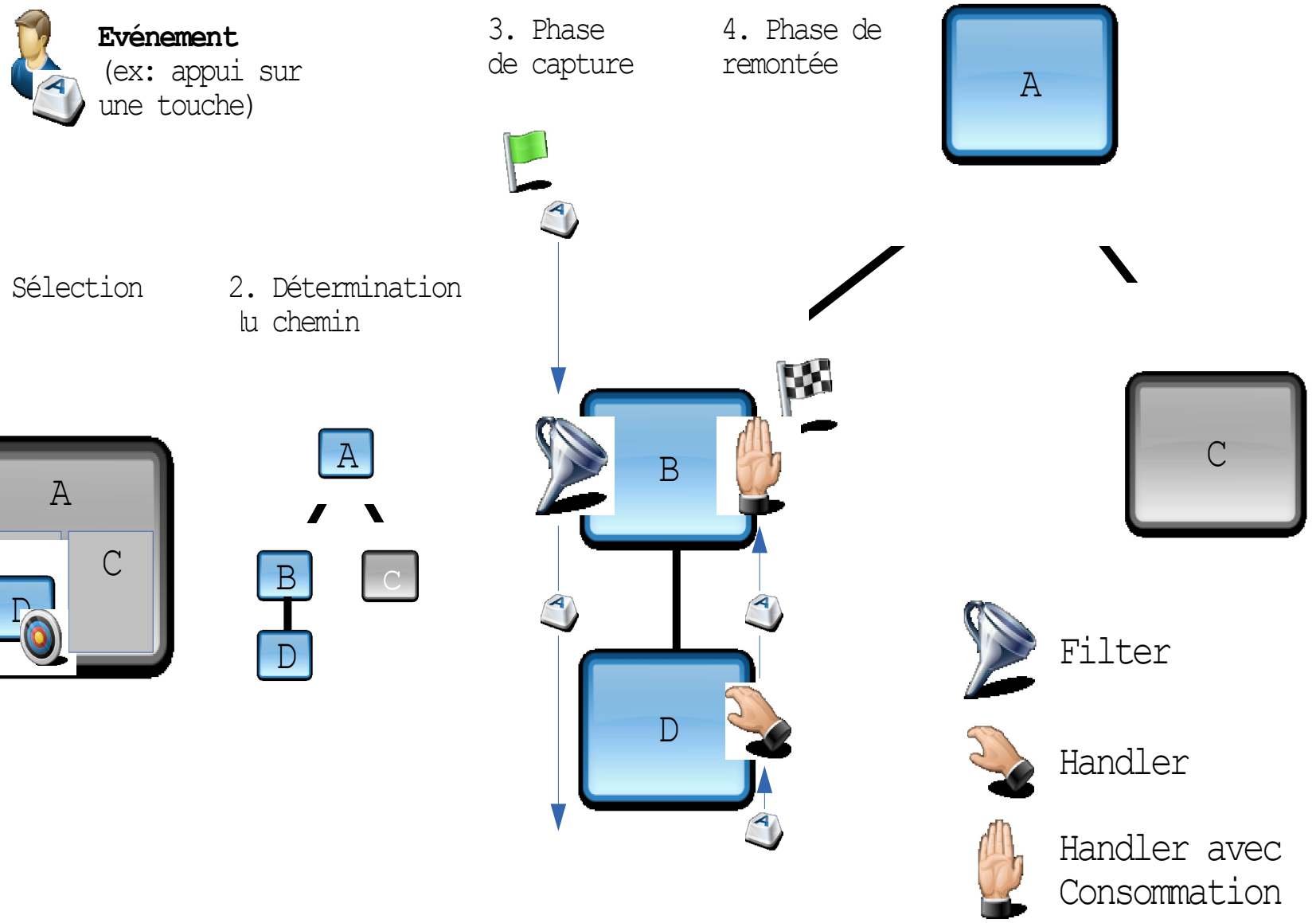
GESTION ÉVÉNEMENTIELLE

Workflow



GESTION ÉVÉNEMENTIELLE

Workflow



GESTION ÉVÉNEMENTIELLE

Utilisation sur un node

```
//Détermination du chemin (surcharger)
public EventDispatchChain buildEventDispatchChain(EventDispatchChain parent)

//Ajout de filtre, pour la phase de capture
public void addEventFilter(EventType<T> type, EventHandler< ? super T>)

//Ajout de gestionnaire, pour la phase de remontée
public void addEventHandler(EventType<T> type, EventHandler< ? super T>)
```

Exemple: ajout d'un listener

```
node.addEventHandler(InputEvent.ANY, new EventHandler<InputEvent> {...});
```

Les composants possèdent des raccourcis pour définir des listeners

```
//sous la forme setOnEvent-type(EventHandler<? super event-class> value)
setOnMouseClicked(EventHandler<MouseEvent> handler)
```

Bubbling ou Filter event?

Choisir la bonne phase

- Les Capture Events permettent de pouvoir stopper la propagation des événements vers les enfants
  - Exemple : pas de support de clic sur un enfant particulier
- Les Capture Events étendent un comportement à l'ensemble de ses enfants
  - Exemple : un EventFilter sur un MouseClick défini sur un GridPane, sera appelé sur chaque clic sur ce GridPane, y compris sur un de ces enfants (bouton, image...)

Bubbling ou Filter Event

Choisir la bonne phase

- Les Bubbling Event permettent de « remonter » des informations vers un parent après une action donnée
- Aucune des deux méthodes est meilleure, elles sont complémentaires!

Evénements personnalisés

Emettre depuis un composant

- L'enveloppe est de type SauvegarderEvent qui étend javafx.event.Event

```
public class SauvegarderEvent extends Event {

    public static final EventType<SauvegarderEvent> DEFAULT =
        new EventType<>("DEFAULT");

    public SauvegarderEvent() {
        super(DEFAULT);
    }
}
```

- On émet l'événement depuis une méthode du composant

```
public void sauvegarder() {
        this.fireEvent(new SauvegarderEvent());
}
```

Evénements personnalisés

Gérer l'écoute

- Ecouter un événement personnalisé depuis le parent sur l'enfant

```
protected void addListenerToChild() {
    this.childComponent.addEventFilter(SauvegarderEvent.DEFAULT,
        new EventHandler<SauvegarderEvent>() {

            @Override
            public void handle(final SauvegarderEvent event) {
                // ...
                // event.consume() ;
            }
        }
    );
}
```

ALTERNATIVE
Listener avec Databinding

Définition d'un EventHandler greffable par Databinding dans le composant

```
private final ObjectProperty<EventHandler<SauvegarderEvent>> onSauvegarderAction =
    new SimpleObjectProperty<EventHandler<SauvegarderEvent>>();

public final ObjectProperty<EventHandler<SauvegarderEvent>> onSauvegarderActionProperty() {
    return this.onSauvegarderAction;
}

public final void setOnSauvegarderAction(final EventHandler<SauvegarderEvent> handler) {
    this.onSauvegarderAction.set(handler);
}

public final EventHandler<SauvegarderEvent> getOnSauvegarderAction() {
    return this.onSauvegarderAction.get();
}
```

ALTERNATIVE
Listener avec Databinding depuis le parent

- En FXML

```
<DetailsExamenBox fx:id="detailsExamenBox"
                  onSauvegarderAction="#sauvegarder"/>
```

- En Java

```
protected void addListener() {
    this.childComponent.onSauvegarderActionProperty().setValue(
        new EventHandler<SauvegarderEvent>() {

            @Override
            public void handle(final SauvegarderEvent sauvegarderEvent) {
                // Code here!
            }
        }
    );
}
```

ALTERNATIVE
Listener avec Databinding

- Emission de l'événement depuis le composant

```
public void dispatch(final SauvegarderEvent zenikaEvent) {
    final EventHandler<SauvegarderEvent> sauvegarderEventEventHandler =
        this.onSauvegarderAction.get();

    if (sauvegarderEventEventHandler != null) {
        sauvegarderEventEventHandler.handle(zenikaEvent);
    }
}
```

- Le binding sur l'événement s'active alors si défini (valeur non nulle) et le listener est appelé
- Cette technique est utilisée dans l'implémentation de nombreux composants JavaFX

BILAN

Les événements locaux

- Deux méthodes sont possibles
- La méthode par databinding est la plus explicite et donc recommandée :
  - Déclaration explicite de l'événement via un ObjectProperty
  - Possibilité de définir le handler en FXML ou Java
- Le dispatch classique peut toutefois être utilisé dans certains cas
  - Evénements locaux « internes » au composant qui ne seront pas écoutés en dehors (exemple : clic sur cellule de tableau qui ne sera pas écouté en dehors du composant tableau)

Gestion du multitouch

Pour plate-formes tactiles

- Support natif dans JavaFX
- Ceux-ci sont gérés de manière similaire aux interactions "classiques" :
  - TouchEvent
  - GestureEvent
  - *ScrollEvent : faire glisser deux doigts, axe vertical*
  - *RotateEvent : rotation des points de contact*
  - *ZoomEvent : "pincer pour zoomer"*
  - *SwipeEvent : faire glisser un doigt, axe horizontal ou vertical*

GESTION DU MULTITOUCH

TouchEvent et GestureEvent

- TouchEvent est bas-niveau
  - Donne le détail des points de contact avec l'écran tactile
- GestureEvent est plus haut niveau
  - Ses sous-classes correspondent aux 4 gestes les plus "classiques"
  - Chaque geste (sauf le swipe) est précédé d'un événement "started" et suivi d'un "finished"
  - Il existe une hierarchie sur les types d'événements

    GestureEvent.ANY > RotateEvent.ANY > RotateEvent.ROTATE_STARTED, RotateEvent.ROTATE, RotateEvent.ROTATE_FINISHED

GESTION DU MULTITOUCH

Exemple

```
// sur un geste zoom, élargir le composant
someComponent.setOnZoom(new EventHandler<ZoomEvent>() {

    public void handle(ZoomEvent event) {
        ((Node) event.getTarget()).setScaleX(event.getTotalZoomFactor());
    }
});
```

- Les interactions de type zoom, défilement sont aisées
  - Support vectoriel natif de JavaFX sur tous les éléments Node
  - Gestion des événements aisée

Gestion du multitouch

Notes particulières

- Les interactions tactiles peuvent donner lieu à des événements non tactiles
  - Exemple : drag + scroll, clic + touch, etc...
  - mouseEvent.isSynthetized() à true si l'événement est tactile
  - gestureEvent.isDirect() à false si l'événement provient d'un touchpad
- Attention donc à ne pas enregistrer des listeners en double!
  - Exemple : listener faisant défiler la page enregistré sur le scroll et le drag = 2x plus de mouvement que prévu

Bus d'événement

Un design pattern utile

- Parfois des événements peuvent intéresser plusieurs composants, à priori deux méthodes sont possibles suivant l'emplacement du composant
  - Utiliser des événements locaux pour « remonter » au composant parent souhaité
  - Utiliser des méthodes publiques sur les composants enfants pour « descendre » jusqu'au composant souhaité

Loi de Demeter

Principe d'encapsulation

- La loi d'encapsulation (Demeter) sur les composants nous incite à ne pas descendre / remonter dans la hiérarchie de composants à plus de un niveau pour ajouter un listener propre
  - Chaque composant du parcourt devrait donc posséder un handler / méthode publique pour router l'action au bon composant
  - Comment rendre le développement plus rapide ?

Bus d'événement

Pourquoi?

- La hiérarchie des composants peut être :
  - Complexe (beaucoup d'embranchements)
  - Longue (beaucoup de niveaux d'enfants / parents)
  - Modifiée durant le développement du projet (évolutions, bugs...)
- En définissant un bus d'événement unique à l'application (singleton), on centralise dans un bean Java l'abonnement / désabonnement à des événements globaux
  - Il devient donc possible d'écouter un événement « global » sans être attaché à l'arbre des composants
  - Fonctionnement de type publish / subscribe

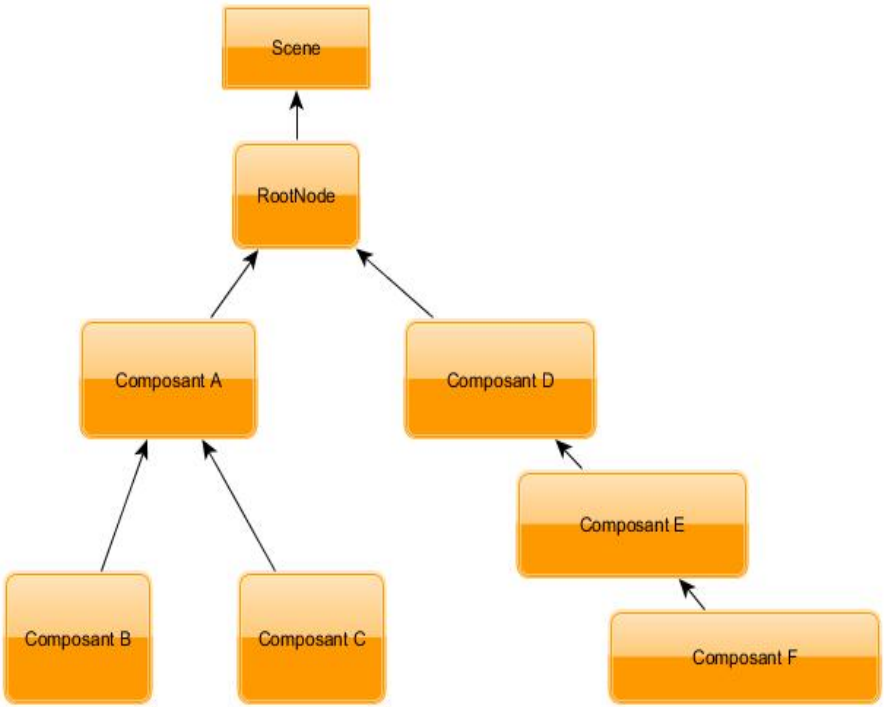
Bus d'événement

Cas d'utilisation

Le composant B veut s'abonner à un événement du composant F

Les composants B, D et F veulent être notifiés d'une déconnexion réseau

Les composants C et E veulent être notifiés d'une fermeture d'une popup modale



Bus d'événement

Implémentations

- Plusieurs frameworks proposent des bus d'événements
  - Spring Events
  - RacpFX
  - Open Dolphin
  - Granite Data Services
- Choix techniques
  - Un bus d'événement peut s'exécuter dans l'Application Thread de JavaFX ou dans un autre géré manuellement
  - Son utilisation peut être fortement couplée à un framework donné