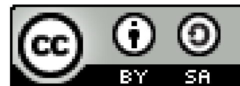
Communication into an Application

Laurent Jouanneau

Course on Mozilla Education and Technologies @ Evry
December 2011

Creative Commons Attribution-Share Alike 3.0 licence
http://creativecommons.org/licenses/by-sa/3.0/



Splitting code into components

- Main idea:
  - Create several component instead of a single big piece of code
  - A component does a single thing, but does it well
  - KISS principle = Keep It Simple and ~~Stupid~~ Smart
  - The component provides or uses mecanism to communicate with other unknown components

Licence CC-by-sa

Splitting code into component, why?

- It's easier to maintain, to debug
- It's easier to do unit tests
- It's easier to make improvements
- It's allow to reuse code
- It's easier to develop the application in a team

Licence CC-by-sa

Splitting code into component, how?

- Don't put everything in few javascript files and xul files.
- For XUL files
  - Use overlays when the XUL interface is complex
  - Use XBL to create interface components
- For the logic code
  - Split the logic into objects in separate JS file, JS modules, XPCOM components
  - The JS scripts of a XUL page is just a glue between these components and the UI

Licence CC-by-sa

Splitting code into component, ex

- Example 1: In a JS function on a click, don't do directly data management + UI change
  - Data management : in a specific component
  - UI change in a dedicated object, which manage window statements etc..
- Example 2: you have a grid of data, with its own behavior, and it could be interesting to reuse it in other windows/projects : create an XBL

Licence CC-by-sa

Decoupling components

- More you have components, more you have interactions between them, more it could be difficult to make changes or to fix bugs, if dependencies are strong
  - Strong dependencies == explicit components call in other components
- Solution = decoupling = using specific design patterns to reduce dependencies, then to reduce explicit call to components
- The most used pattern : listeners and broadcasters
  - Listener : an object registered into a component, and called by this component during its execution
  - Broadcaster : a component owning listeners and calling by other components to notify listeners

Licence CC-by-sa

Listeners into Mozilla

- For the UI
  - DOM Event Listeners
  - XUL Broadcaster/observers
  - XUL Commands
- For the logic code
  - Any listener implementation
  - The Observer Service

Licence CC-by-sa

DOM Event Listeners

- An event is an object, a piece of information, which is passed to an element and its parents. There are 3 phases:
    1) capture: from the root to the target element
    2) at-target: on the target element
    3) bubble: from the target to the root element
- We can react to an event, by registering a listener on a element which will receive this event
- Elements have default behaviors on some events, we can canceled them.
- On event objects, some properties: target, currentTarget, bubbles, cancelable
- Some methods: stopPropagation(), preventDefault()

Licence CC-by-sa

DOM Event Listeners

- Registering an event listener

```
<box oncommand="..js..code..." />

Function myListener(event) { ...js..code.. }

elt.oncommand = myListener;

elt.addEventListener("command", myListener, false);
elt.removeEventListener("command", myListener, false);
```

- With AddEventListener you can add several listeners for the same event on the same element. You can also indicate if the listener is called during the capture phase (true) or during the bubble phase (false)

Licence CC-by-sa

DOM Event Listeners

- Creating events :

```
function simulateClick() {

  var evt = document.createEvent("MouseEvents");

  evt.initMouseEvent("click", true, true, window,
    0, 0, 0, 0, 0, false, false, false, false, 0, null);

  var cb = document.getElementById("checkbox");
  var canceled = !cb.dispatchEvent(evt);
  if(canceled) {
    // A handler called preventDefault
    alert("canceled");
  } else {
    // None of the handlers called preventDefault
    alert("not canceled");
  }
}
```

Licence CC-by-sa

XUL Broadcasters/observers

- Broadcasters are elements that hold some attributes shared by other elements called « observers », i.e., all attributes set or modify on a <broadcaster> element, are forwarding on elements that observe this broadcaster
- An element observes a broadcaster by indicating the id of the broadcaster into an « observes » attribute

```
<broadcasters>
  <broadcaster id="offline_command" label="Offline" accesskey="f"/>
</broadcasters>

<keyset>
  <key id="goonline_key" observes="offline_command" modifiers="accel" key="O"/>
</keyset>

<menuitem id="offline_menuitem" observes="offline_command"/>

<toolbarbutton id="offline_toolbarbutton" observes="offline_command"/>
```

Licence CC-by-sa

XUL Broadcasters/observers

- `document.getElementById('offline_command').setAttribute('disabled','true');`

  In this example, all elements observing the broadcaster will be disabled.

- To observe a specific attribute, and/or to know when the broadcast is triggered:

```
<broadcasterset>
  <broadcaster id="colorChanger" style="color: black"/>
</broadcasterset>

<button label="Test">
  <observes element="colorChanger" attribute="style"
            onbroadcast="alert('Color changed');"/>
</button>

<button label="Observer"
  oncommand="document.getElementById('colorChanger').setAttribute('style','color:red');"
/>
```

Licence CC-by-sa

XUL Commands

- The <command> element is like a broadcaster, dedicated to actions

```
<command id="cmd_openhelp" oncommand="alert('Help!');" label="Help"/>

<button command="cmd_openhelp"/>

<menuitem command="cmd_openhelp"/>

<key  command="cmd_openhelp" modifiers="alt" key="H"/>


<button label="Disable"
        oncommand="document.getElementById('cmd_openhelp')
                           .setAttribute('disabled','true');"/>
<button label="Enable"
        oncommand="document.getElementById('cmd_openhelp')
                           .removeAttribute('disabled');"/>
```

Licence CC-by-sa

Architecture with listeners

Licence CC-by-sa

Example: naive implementation

Many defects with this implementation

Licence CC-by-sa

Example: naive implementation

- Many defects with this implementation
  1) If the UI change: we have to change all functions
  2) If the storage data change: we have to change all functions
  3) If we add a new operation: we certainly have to duplicate code, and this is another function to change when 1) and 2)

Licence CC-by-sa

Example: better implementation

UI

records table

Add

Remove

UITableManager

remove()

add()

update()

dataManager

this.data = []

remove()

add()

getData()

But not the best. How an other component can react on data change?

Licence CC-by-sa

Example: best implementation

- Solution: decoupling.
- Using
  - listener/callbacks
  - Or the observer service

Licence CC-by-sa

Example: best implementation

With the observer service
Of Gecko

Licence CC-by-sa

The observer service

- Creating an observer

```
var myObserver = {
  observe: function (aSubject, aTopic, aData) {
  }
}
```

  - subject: an object, often the object related to the message or the notifier
  - Topic: the message name
  - Data: additional data, parameters...

- Registering an observer : use the Services module

```
Components.utils.import("resource://gre/modules/Services.jsm");
Services.obs.addObserver(myObserver, "my-data-changed", false);
```

Licence CC-by-sa

The observer service

- Notifying observers

```
Services.obs.notifyObservers(dataManager, "my-data-changed", "");
```

- Allow communication between any kind of components: XBL, JS Objects, JS modules, XPCOM...
- Note the old way to retrieve the observer components

```
var obs = Components.classes["@mozilla.org/observer-service;1"]
            .getService(Components.interfaces.nsIObserverService);
```

Licence CC-by-sa

Example: best implementation #2

Licence CC-by-sa

Specific listeners

```
var dataManager = {
   _listeners : [],
   addListener: function (listener) {
     this._listeners.push(listener);
   },

   _notify: function() {
    this._listeners.forEach(
       function(elt, idx, thearray){
         elt.onDataChanged(this);
       }, this);
   },


   remove: function (id) {
     this._notify();
   }
}
```

```
var myListener ={
   onDataChanged: function(manager){

   }
}
```

```
dataManager.addListener(myListener);
```

Licence CC-by-sa