XUL & Gecko Commands

Laurent Jouanneau
Course on Mozilla Education and Technologies @ Evry
December, 2011

XUL commands

Implementation

- In a DOM Event listener for the « command » event, like any DOM Event listener
- It can be a Javascript function

```
<button id="mybutton"/>

function myCommand(event) { alert('hello'); }

document.getElementById('mybutton')
        .addEventListener('command', myCommand, false);
```

- or it can be a script in the « oncommand » attribute

```
<button oncommand="alert('Hello!');"/>
```

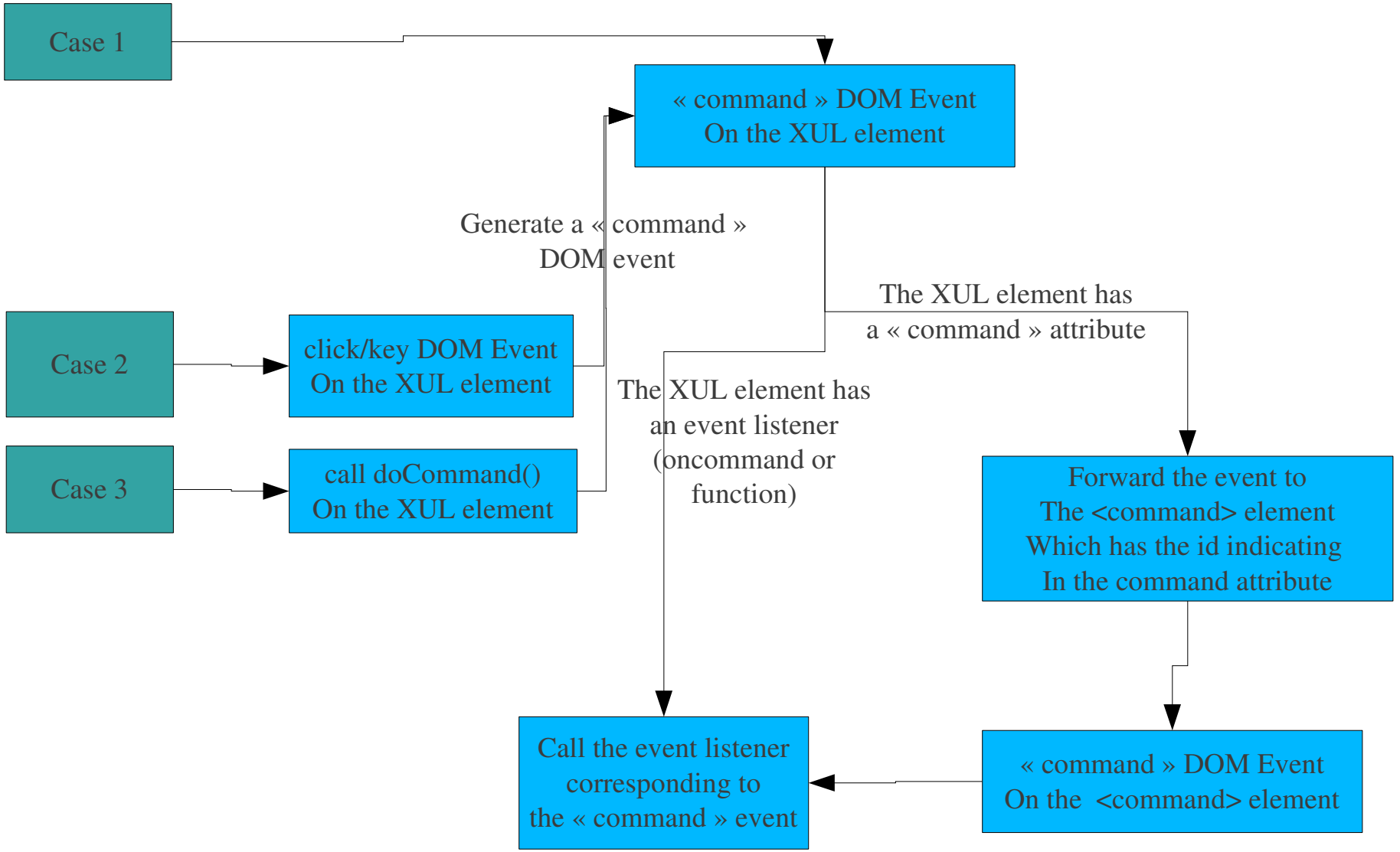
The <command> element

- A <command> element holds the command event listener
- It is a broadcaster: any attribute and any changes on this attribute are propagated to any XUL element observing it
- A XUL element observes a command via the « command attribute », indicating the id of the <command> element.

```
<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"/>
```

- <command> elements should be inside a <commandset> element

Invocation



Enabling / Disabling

- Just set to true or remove the « disabled » attribute on the <command> element. If set, all xul elements observing the command will be disabled.

```
<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');"/>
```

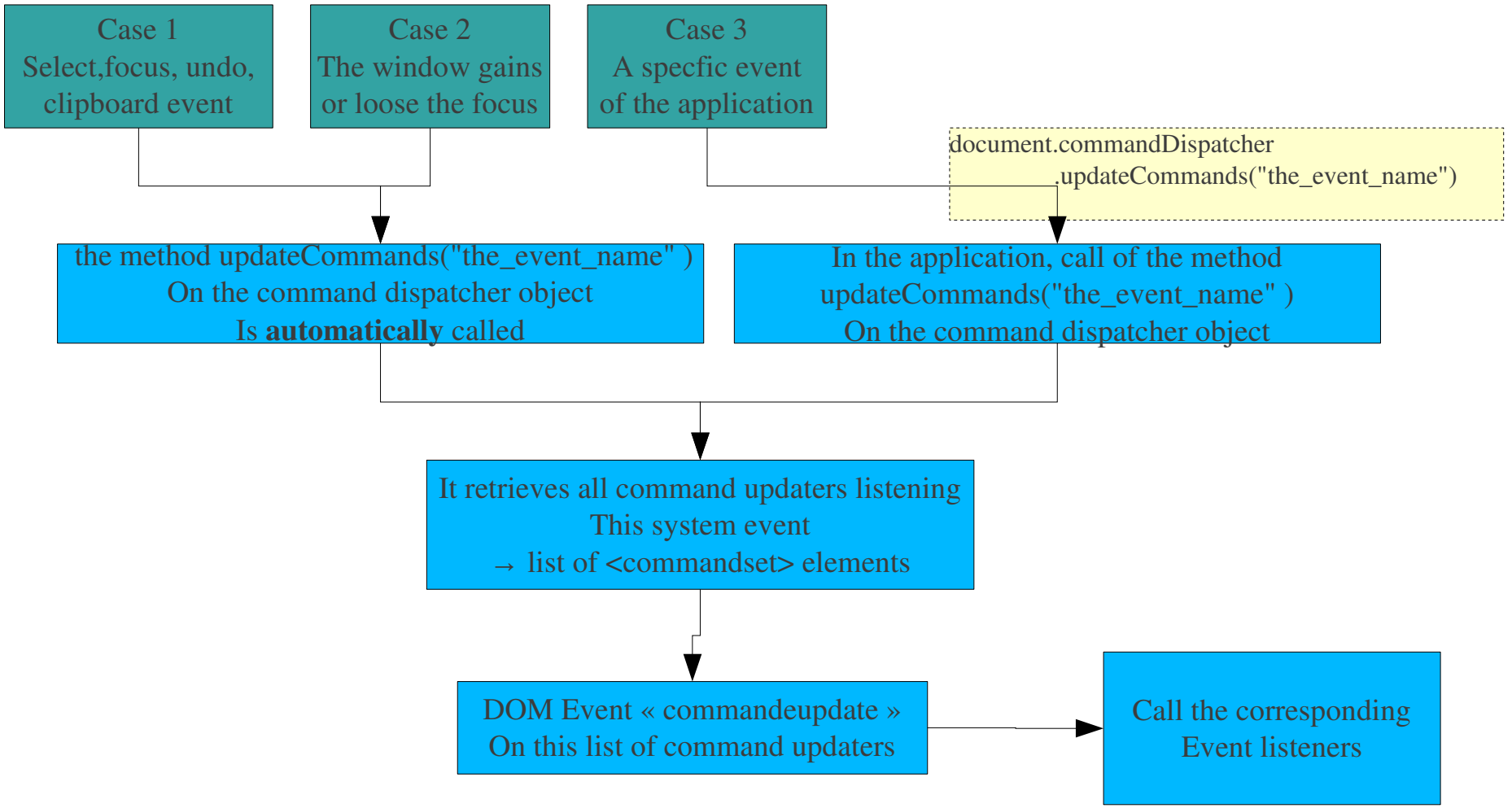
Commands updating

- Goal: change or update the status of some xul commands, when some (non DOM) events occur.
- Example: in a textbox/textarea/editor, when the user put the focus on it, and select some text, the cmd_paste command (and others like cmd_copy, cmd_delete...) should be enabled. So, when the system event « select » and « focus » occur, we have to enable these commands, otherwise, we should disable them.
- Example: we have to do other things automatically when we want to disable/enable a command
- Solution: using a command updater

A command updater

- This is a <commandset> with:
  - An attribute commandupdater="true"
  - An attribute events="", containing the list of events during which the update should occurs : « select », « focus », « clipboard », « undo ». You can imagine your own events.
  - A DOM Event listener on the « commandupdate » event , attached to it (a function or a oncommandeupdate attribute).
- The event listener on the « commandupdate » event, can then modify some <command> elements or do other changes

How it is working



Gecko commands

commands system of Gecko

- This is a an other command system, not based on XUL elements like <command>.
- However we can use it with XUL commands. A command can be implemented using the Gecko commands system, and XUL commands are used to « link » them to the interface.
- Advantages:
  - Independant of a user interface
  - XBL, XPCOM and JSM components can implement some commands
- Many internal commands are implemented using this system. Ex: commands cut/copy/paste/delete in textboxes.

Implementation

- A command is implemented in an object called a « controller », following the nsIController interface
- A controller can implement several command at a time
- Methods to implement:
  - DoCommand: execute the command itself
  - IsCommandEnabled: return true if the command is enable
  - SupportsCommand: indicate if the controller supports the given command
  - OnEvent: should execute code corresponding to the given event name (non DOM event)

Implementation #2

- Example: this controller supports two commands

```
var myController = {
    doCommand : function(command) {
        if (command == 'cmd_mycommand1') {
            //..
        }
        else if (command == 'cmd_mycommand2') {
            //..
        }
    },
    isCommandEnabled : function (command) {
        return true;
    },
    supportsCommand : function (command) {
        if (command == 'cmd_mycommand1' ||
            command == 'cmd_mycommand2')
            return true;
        return false
    },
    onEvent: function (eventName) {
        if (eventName == 'focus') { /*...*/ }
    }
}
```

Declaring the controller

- A controller should be added in the list of controllers of the window, or of a XUL Element which can have the focus
- A list of controllers is a nsIControllers object, in the « controllers » property of « window » or a xul element
- Methods:

```
appendController(controller)
insertControllerAt(index, controller)
removeController(controller)
removeControllerAt(index)

getControllerAt(index)
getControllerCount()
getControllerForCommand(command)
```

- Example:

```
window.controllers.appendController(myController);

myXulElement.controllers.appendController(myController);
```

Declaring a controller #2

- Tips: inside a component, to retrieve the window, use the window mediator
- The window mediator allows to retrieve a specific window or the list of all opened windows.

```
var windowMediator = Components.classes["@mozilla.org/appshell/window-mediator;1"]
                               .getService(Components.interfaces.nsIWindowMediator);


// mywindowtype is the value of the « type » attribute on the xul <window> element
var myWindow = windowMediator.getMostRecentWindow("mywindowtype");

myWindow.controllers.appendController(myController);
```

The command dispatcher

- It is an object (nsIDOMXULCommandDispatcher) allowing:
  - To move the focus inside the window or to one of the opened window of the application
  - To retrieve the focused element
  - To update commands
  - To retrieve a controller corresponding to a specific command or the list of controllers
- To retrieve it: document.commandDispatcher

Executing a command

- Typical code:

```
var controller = document.commandDispatcher
                         .getControllerForCommand("cmd_mycommand");

if (controller && controller.isCommandEnabled("cmd_mycommand"))
   controller.doCommand("cmd_mycommand");
```

- A such function is provided by globalOverlay.js: goDoCommand()

```
<script type="application/javascript"
        src="chrome://global/content/globalOverlay.js"/>

goDoCommand("cmd_mycommand");
```

- With a XUL Command:

```
<command id="cmd_mycommand" oncommand="goDoCommand('cmd_mycommand')"/>
```