Mobile Front Controller

An overview

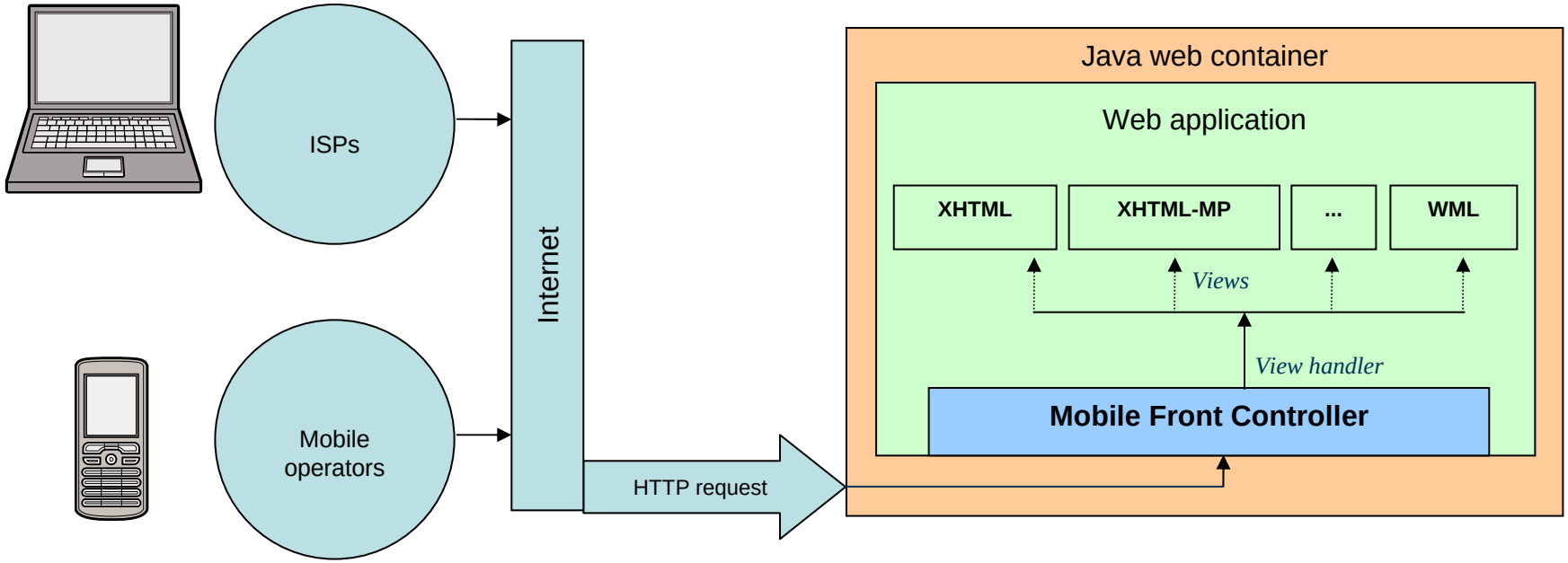
ERICSSON

TAKING YOU FORWARD

Mobile Front Controller

Mobile Front Controller (MFC) is a light-weight web application framework that:

- Selects and detects views (subdirectories).
- Shares logic between views using action commands.



Other frameworks can be used together with MFC.

Mobile Front Controller - components

- Controller servlet (`ControllerServlet`)
  - Makes sure that a view is selected (using a view handler).
  - Executes action commands and forwards/redirects to, for example, a JSP page.
  - Specified in the web application deployment descriptor: `WEB-INF/web.xml`:

```
<servlet>
  <servlet-name>ControllerServlet</servlet-name>
  <servlet-class>
  webframework.mobilefrontcontroller.controller.ControllerServlet
  </servlet-class>
  <load-on-startup>3</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>ControllerServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

    In the example above, the URL pattern is set to `*.do`,
    which means that action commands are executed by calling,
    for example, `http://localhost:8080/MFCApplication/TestAction.do`

- Interfaces
  - Action commands: `ActionCommand`, `DispatchActionCommand`
  - View handlers: `ViewHandler`

Action commands

- Action command, a class that implements the `ActionCommand` interface, which has an execute method for performing logic.
- The locations of action commands are specified in `WEB-INF/mobilefrontcontroller.xml`:

```
<mobile-front-controller
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="mobilefrontcontroller_1_1.xsd">

  <!– Specific action command mapped from an action name to a class name -->
  <action-command-mapping>
    <action-name>Test</action-name>
    <action-class>example.test.TestCommand</action-class>
  </action-command-mapping>

  <!-- Action command package paths, where to look up other
       action commands.-->
  <action-command-package-path>example.actions</action-command-package-path>

  <!-- … other configuration … -->
</mobile-front-controller>
```

Action commands – interfaces

An action commands implements:

- the `ActionCommand` interface (required):

```
public interface ActionCommand {
    public String execute(
                HttpServletRequest request,
                HttpServletResponse response);
}
```

- the `DispatchActionCommand` interface (optional):

```
public interface DispatchActionCommand {
    public DispatchType getDispatchType();
}
public enum DispatchType {
    FORWARD,
    REDIRECT;
}
```

Note: Dispatch type defaults to `DispatchType.FORWARD` unless the `DispatchActionCommand` interface is implemented.

Action commands – example 1

An action command that only forwards to a JSP page.

```
package examples.mobilefrontcontroller.actions;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import
   webframework.mobilefrontcontroller.actions.ActionCommand;

public class TestCommand implements ActionCommand {
    public String execute(HttpServletRequest request,
                          HttpServletResponse response) {
        return "test.jsp";
    }
}
```

Action commands – example 2

An action command that invalidates a session and redirects to a JSP page.

```
package examples.mobilefrontcontroller.actions;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import webframework.mobilefrontcontroller.actions.ActionCommand;
import webframework.mobilefrontcontroller.actions.DispatchActionCommand;
import webframework.mobilefrontcontroller.actions.DispatchType;

public class Logout implements ActionCommand, DispatchActionCommand {

    public String execute(HttpServletRequest request,
                          HttpServletResponse response) {
        request.getSession().invalidate();
        return "index.jsp";
    }

    public DispatchType getDispatchType() {
        return DispatchType.REDIRECT;
    }
}
```

Action commands – example 3

An action command that writes to the response object and returns null. Neither a forward or a redirect is performed.

```
package examples.mobilefrontcontroller.actions;
// ...
import webframework.mobilefrontcontroller.actions.ActionCommand;

public class Output implements ActionCommand {
    public String execute(HttpServletRequest request,
                          HttpServletResponse response) {
        PrintWriter writer = null;
        try {
            response.setContentType("text/plain");
            writer = response.getWriter();
            writer.println("Hello world!");
            writer.flush();
        } catch (IOException ex) { } finally {
            if (writer != null) writer.close();
        }
        return null;
    }
}
```

Views

- A view is a subdirectory in a web application.
- Example of views:
  `http://localhost:8080/MFCApplication`
  - `/xhtml/`
  - `/xhtmlmp/`
  - `/wml/`
  - `/devices/sonyericsson/`
  - `/devices/iphone/`
- A view is represented by the `View` class:

```
public final class View {
   // ...
    public View(String directoryPath) {...}
    public String getDirectoryPath() {...}
}
```

- A view is detected and selected by a view handler...

View handlers

- A view handler is a class that implements the `ViewHandler` interface.
- A view handler can:
  - Detect if a request (action command) comes from a view or not.
  - Detect and select a view.
- A default view handler, `DefaultViewHandler`, is used unless a custom view handler is specified. The default view handler supports the views: `xhtml`, `xhtmlmp`, `wml`.

View handlers - interface

```
public interface ViewHandler {
   void init(ServletConfig servletConfig);
   boolean isViewSelected(HttpServletRequest request);
   View selectView(HttpServletRequest request);
}
```

- `void init(ServletConfig servletConfig)`
  is called when the controller servlet is initialized.
- `boolean isViewSelected(HttpServletRequest request)`
  is used by the controller servlet to check whether a request (action command) comes from a view or not.
- `View selectView(HttpServletRequest request)`
  is used by the controller servlet to detect and select a view.

View handlers - customization

- A custom view handler can be created by implementing the `ViewHandler` interface.

- The customized view handler is specified in `WEB-INF/mobilefrontcontroller.xml`:

```
<mobile-front-controller
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="mobilefrontcontroller_1_1.xsd">
  <!-- … other configuration … -->
  <parameters>
    <view-handler>example.CustomViewHandler</view-handler>
  </parameters>
</mobile-front-controller>
```


ERICSSON
TAKING YOU FORWARD