Status: Draft Version 0.5

Web Applications Blueprints



© Bosch Software Innovations GmbH, 2011. All rights reserved. Dissemination or reproduction of this publication or any part of it for any purpose or in any form whatever is not permitted without the prior express written consent of Bosch Software Innovations GmbH. Information contained in this publication may be subject to revision without advance notice. MLDS, Visual Rules and Work Frame Relations are registered trademarks of Bosch Software Innovations GmbH. BOSCH and the symbol are registered trademarks of Robert Bosch GmbH, Germany. Some of the product and company names used in this white paper are trademarks and/or registered trademarks. They are used explicitly for reference purposes and are, independent of marking, property of their respective owners.

[1 Introduction 4](#_Toc294284232)

[1.1 Context 4](#_Toc294284233)

[1.2 Document’s topic 4](#_Toc294284234)

[2 webTop 5](#_Toc294284235)

[2.1 Introduction 5](#_Toc294284236)

[2.2 Features 5](#_Toc294284237)

[2.2.1 Web remoting 5](#_Toc294284238)

[2.2.2 Multi-device rich UI management 6](#_Toc294284239)

[2.2.3 Common Features 11](#_Toc294284240)

[2.3 Security integration 11](#_Toc294284241)

[2.4 Portal integration 12](#_Toc294284242)

[2.5 Web transfers performance 13](#_Toc294284243)

[2.6 Platform administration 14](#_Toc294284244)

[2.7 Architecture 14](#_Toc294284245)

[2.8 Integration patterns 15](#_Toc294284246)

[2.9 Use cases 17](#_Toc294284247)

[2.9.1 eMobility and WCMS integration 17](#_Toc294284248)

[2.9.2 FormFlow Platform 18](#_Toc294284249)

[3 WebViews (This is still work in progress) 20](#_Toc294284250)

[3.1 Introduction 20](#_Toc294284251)

[3.2 UI Designer 20](#_Toc294284252)

[3.3 Architecture 21](#_Toc294284253)

[3.4 Integration patterns 22](#_Toc294284254)

[3.5 Use cases 22](#_Toc294284255)

[3.5.1 FormFlow Platform 22](#_Toc294284256)

# Introduction

## Context

Building web applications that enables richer User Experience has become quite complex. This is partly due to the multiplication of technologies that may be used to achieve that goal.

To address this complexity and enhance project teams’ productivity, Bosch SI proposes a **mastered** **Java-based framework** to build Rich Internet Applications.

Built by experienced web application developers, this framework is an expression of Bosch’s internal savoir-faire in the area. Hereby, it uses a set of selected new technologies that proved to be reliable over the past few years.

Furthermore, its core components were designed to be:

* **Flexible**: they can either be used separately - to enable specific web-related features - or they can be wired together to build a fully functional web applications,
* **Extendable**: they can be extended to provide ad-hoc additional features,
* **Generic**: they only use generic technologies that do not require any environment customization - build and runtime phase alike.

Feature-wise, Bosch SI web application framework also provides support to build solutions addressing the most common technical concerns that need to be implemented:

* **Remoting**: support to quickly expose business services to the Web (Rest, WS-\*, Direct Web Remoting…)
* **Security**: support of secured communication protocols and connection to an external (or built-in) AAA provider.
* **Rich content delivery**: support to generate Rich User Interfaces - depending on the targeted client device - and to optimize client-server data transfers.

## Document’s topic

This document aims to describe Bosch SI MSF[[1]](#footnote-2) components used to build Rich Internet Applications.

To that end, it provides a detailed explanation of the underlying distributed **libraries**, **solutions** and **integration patterns**.

# webTop

## Introduction

webTop is Bosch SI web applications framework’s **central component** and can be described as a “**Business-To-Web springboard**” lightweight platform.

Its main feature is indeed to **effortlessly expose business services to the Web**, thus transforming web clients into business desktops. Additionally, it provides support for web-based features such as messaging and session-management (login/logout).

webTop can also be considered as a **scalable and manageable web platform** to expose business services. All of this platform’s **distributed features** are listed in the following part.

## Features

#### Web remoting

Web remoting is the most important feature delivered by webTop. Two kinds of expositions can be enabled for business services:

* **Direct Web Remoting**
  + Optimized data transfers to Javascript clients
  + Embedded server push technology for data streams
  + Direct exposure of any service that declares beans or primitives as parameters and returned value types
* **RESTful web services**
  + Most generic exposition (native web clients supported communication protocol)
  + Service consumers technologies independent
  + Cacheable responses for better performances

The webTop runtime platform can connect to **any kind of external services** **providing a Java interface**. Supported protocols, among others, are **RMI**, **HTTP** and **WS-\***.

However, **services implementations can also be imported** to webTop, allowing business logic **to be processed locally**.

Additionally, **services proxies can be defined in Java** within the platform. These proxies may wrap the logic of expositions for better Web integration (e.g. exposing services in a RESTful logic).

The following diagram provides a summary of how business services can be exposed through   
webTop:

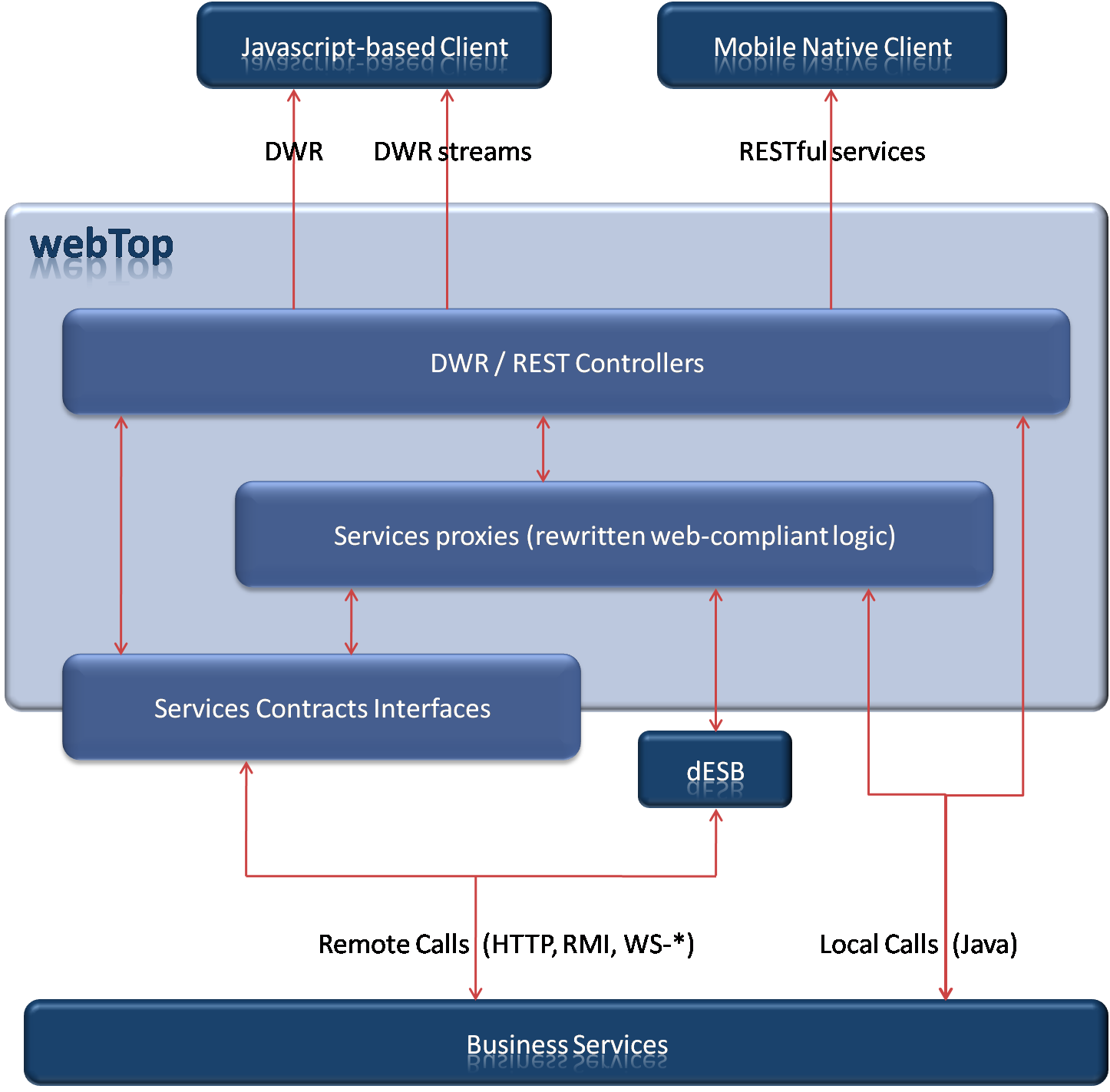


Figure : webTop Remoting Feature

Note: Services configuration can rely on Spring - especially for its powerful remoting API.

#### Multi-device rich UI management

The second central built-in feature of the webTop platform is the **management of a customizable rich web client**.

This client can be mapped to different types of workspaces, each of them with their own set of features and their own delivered user experience. The mapping of the client to one workspace will mainly rely on the following considerations:

* Which is the **targeted device**? (Desktop, Phone, Tablet…)
* What is the **targeted audience**? (Administrator, Customer…)
* What are the **available views** for the current client user and how are they bound to each other?

Fundamentally, these workspaces are **business views containers**.Theydefine the navigation between these views and the way these are displayed within the UI. The following is a description of these different workspaces views.

##### Traditional View

The **Traditional View** handles **browsing** **between views from a menu**, i.e. one selected view is displayed at a time within a “body”.

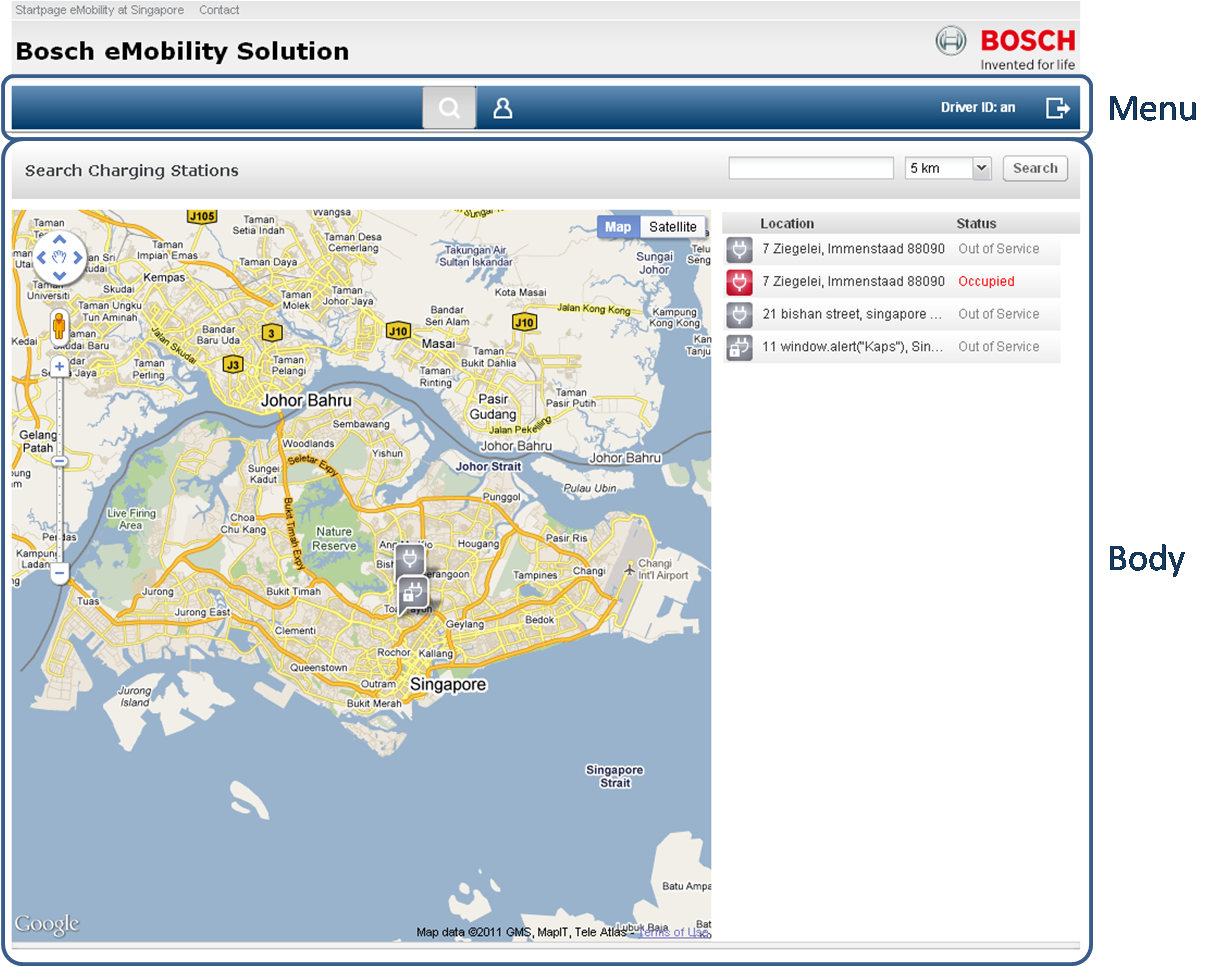


Figure : webTop Traditional View

##### Desktop View

In order to expose views **as dashlets or within dynamic windows**, a workspace must be configured as a **Desktop View** which will have, by default, an **OS / dashboard look and feel**.

Within this container view, the dashlets can be reordered, collapsed or expanded:

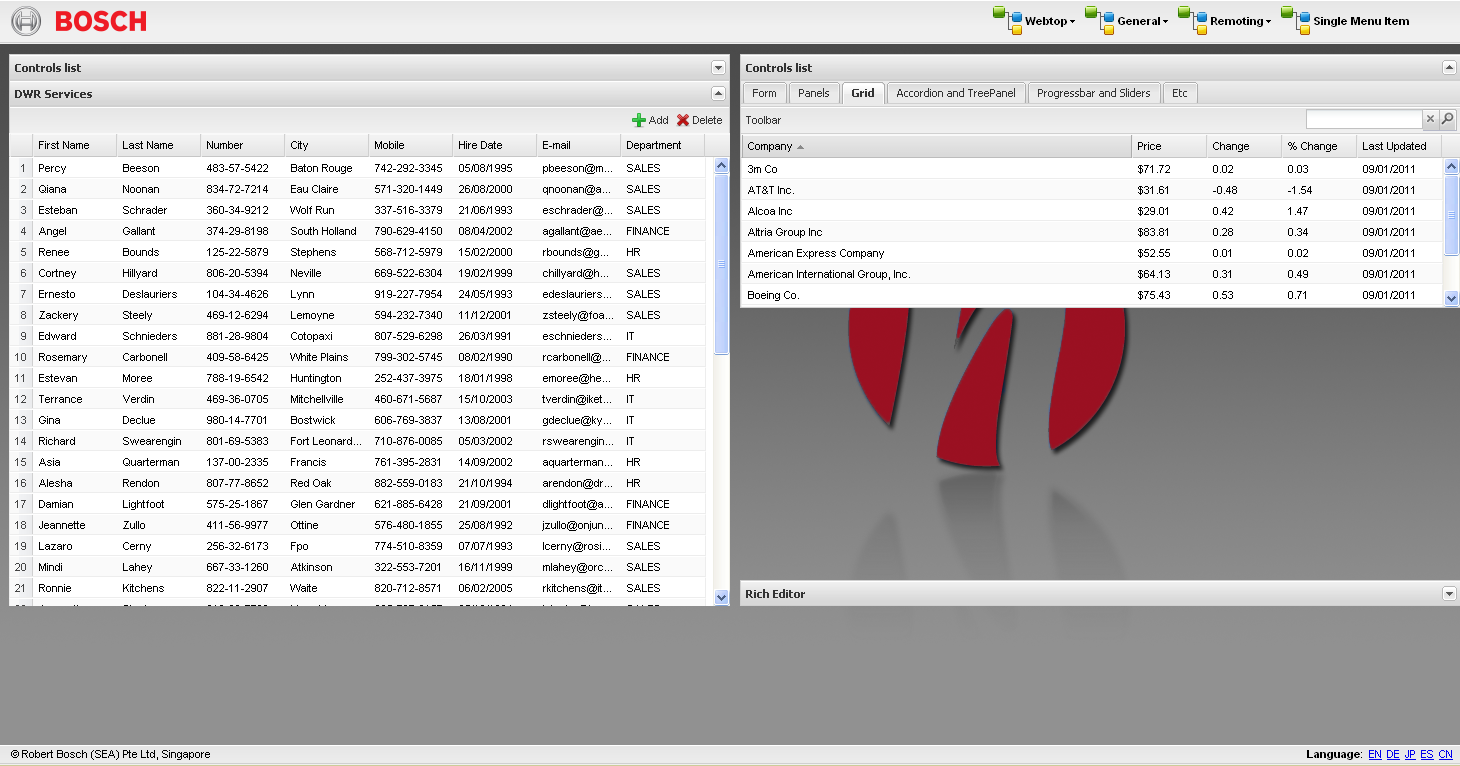


Figure : webTop Desktop View - dashlets

And windows can be invoked from a toolbar and minimized / maximized just like in an OS environment.

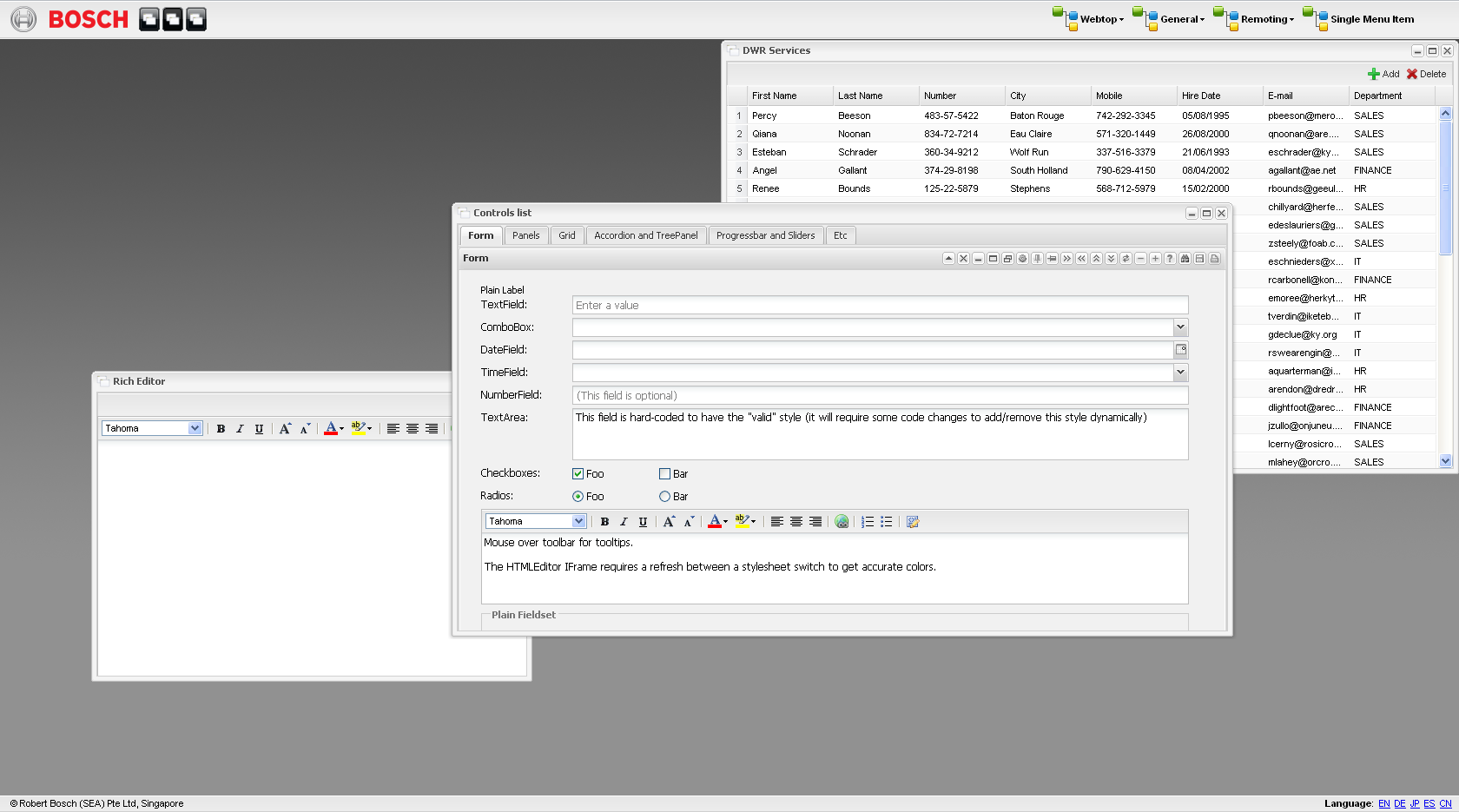


Figure : webTop Desktop View – Windows

Both dashlets and windows may be used at the same time for complex desktops behavior:

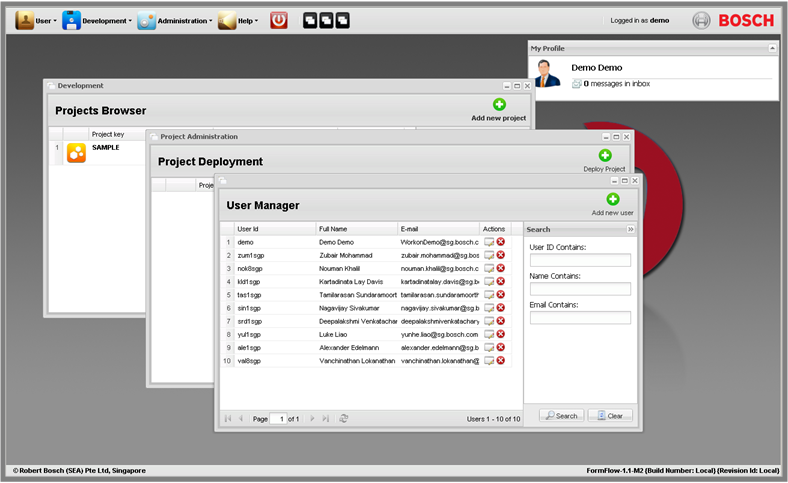


Figure : webTop Desktop View - Misc

##### Isolated View

An **Isolated** **View** can also be generated to display a unique view, targeted by a generated URL (example: a login form view can be available at a webTop generated URL <webTop-workspace-url>/view/login-form). This kind of exposition is convenient for accessing views within standalone windows or frames.



Figure : webTop Isolated Views

##### Mobile App View

Still being engineered, the **Mobile App View** will handle views that can be browsed within small-sized frames.



Figure : webTop - Mobile (Work In Progress)

##### Behind the scenes

In order to render these dynamic user interfaces in a browser, **webTop provides all required resources**: HTML pages, Javascript and CSS files, images…

More precisely, all UI resources are accessible through URLs bound to the following webTop controllers:

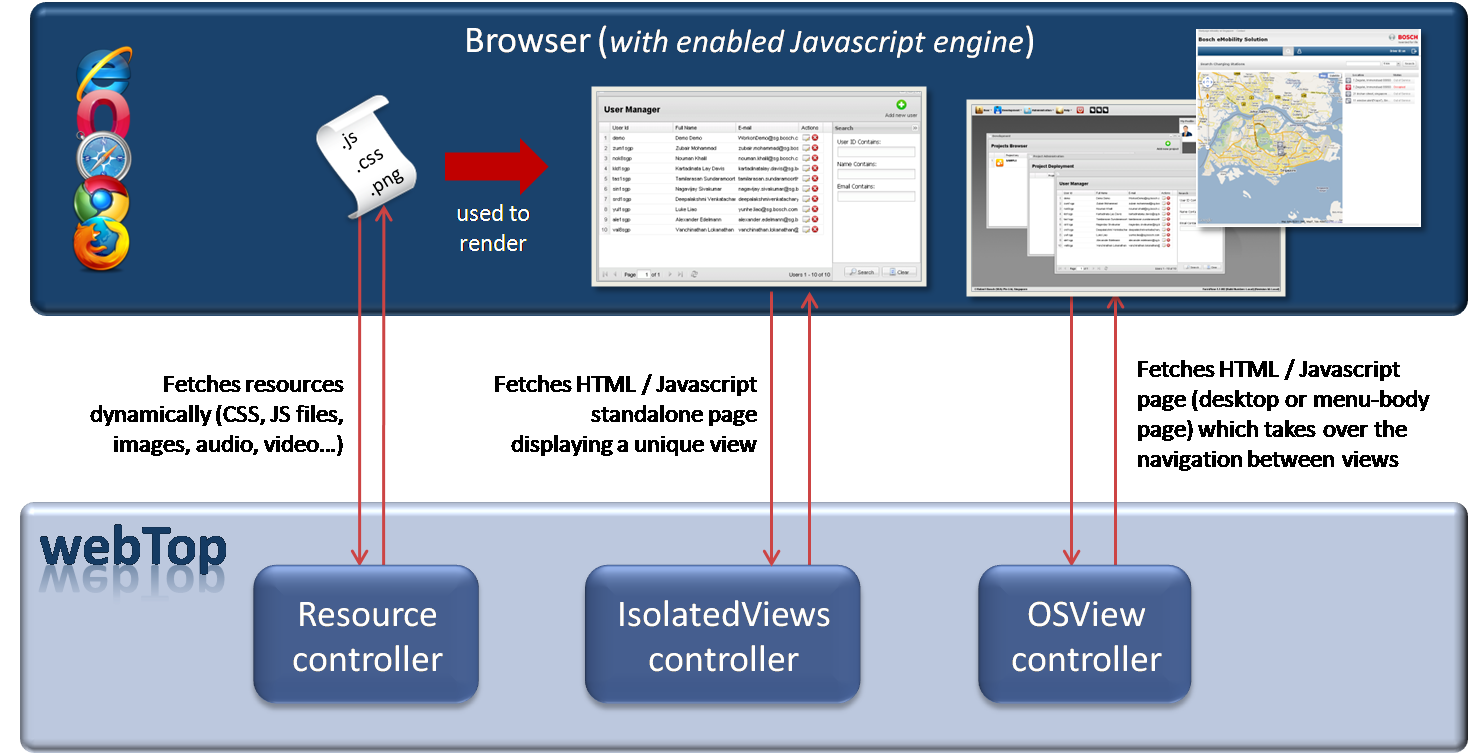


Figure : webTop UI Feature

#### Common Features

There are a number of common features that all webTop views may easily access.

Firstly, each view can embed localized content. Besides, **internationalization** being managed by the platform itself for each client session, the locale setting can be synchronized between views.

Furthermore, each webTop workspace is built on a **front-end messaging micro-platform**. Connected by default to this micro-platform, **views can communicate with each other** if they are displayed at the same time. **Server Pushed Messages** are also directly published to the micro-platform so that any view can access them if needed.

Since webTop supports “cookies and GET parameters”-based navigations between views, **a context can also be passed from a view to another** even if they are not displayed at the same time (e.g. using Isolated Views).

Besides, **a view can generate its own content depending on how it is queried**. For example in an HR-related UI, a view displaying the detailed information of a single Employee would generate its own content based on which employee is targeted.

As for future webTop common features, **accessibility support** (themes, accessible UI patterns, audio content fallback) is under study.

## Security integration

webTop does not declare its own embedded security manager but can integrate with a number of SSO handlers such as:

* **Central Authentication Service**: a standalone service that may run on an external CAS server and connect to LDAPs or databases to validate authentications,
* **Spring Security**: a security framework that can connect to any AAA provider (e.g. **Identity Manager**), regardless of their implementation.

In both cases, security is not fully managed by the platform itself but by wrapping proxies. When an unauthenticated call to the platform is detected, these proxies activate the authentication process (e.g. login form handling, call to AAA provider…).

webTop supports SSL-based connections over HTTP (**HTTPS**) including the use of **secure cookies**.

The platform security integration is described in the diagram below:

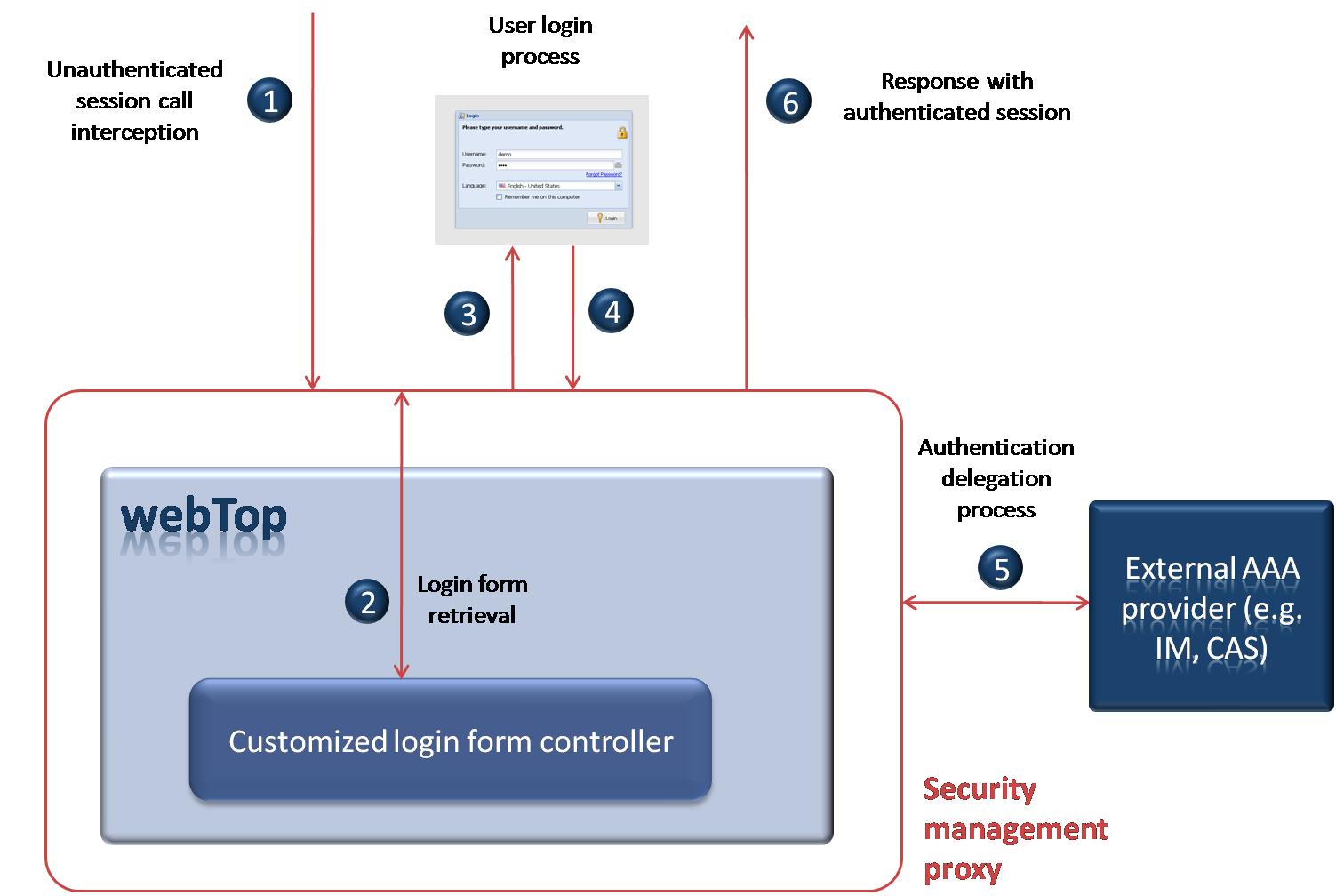


Figure : webTop Security Integration

Furthermore, it is **up to the web application developer to provide the logic to display/hide the views depending on user authorizations**. This can be done by overriding webTop’s *ApplicationViewHandler* and rewriting the logic of the method *isViewValid(viewId)*. Depending on the method’s returned *boolean* value, the targeted view will or will not be accessible to the user.

Note: A webTop Spring Security login form is available by default at the URL*<webTop-platform-url>/login.*

This form can be used by adding these lines to Spring Security configuration file:

<security:form-login login-page=*"/login"* default-target-url=*"/"* authentication-failure-url=*"/login?e=auth"* />

<security:logout logout-url=*"/logout"* logout-success-url=*"/login?e=logout"* />

## Portal integration

Currently, portlets can be integrated in webTop in **iFrames nested in Isolated Views**. webTop’s responsibility in this case would be limited to that of a wrapper’s. If no wrapping is needed, webTop can also generate a web page redirecting to the portlet external URL.

To integrate a webTop view within a portal, this view should be targeted by its isolated URL (*<webTop-url>/view/name-of-the-view*) to be displayed in an **iFrame within a portlet**. The following screenshot provides a live example of such integration.

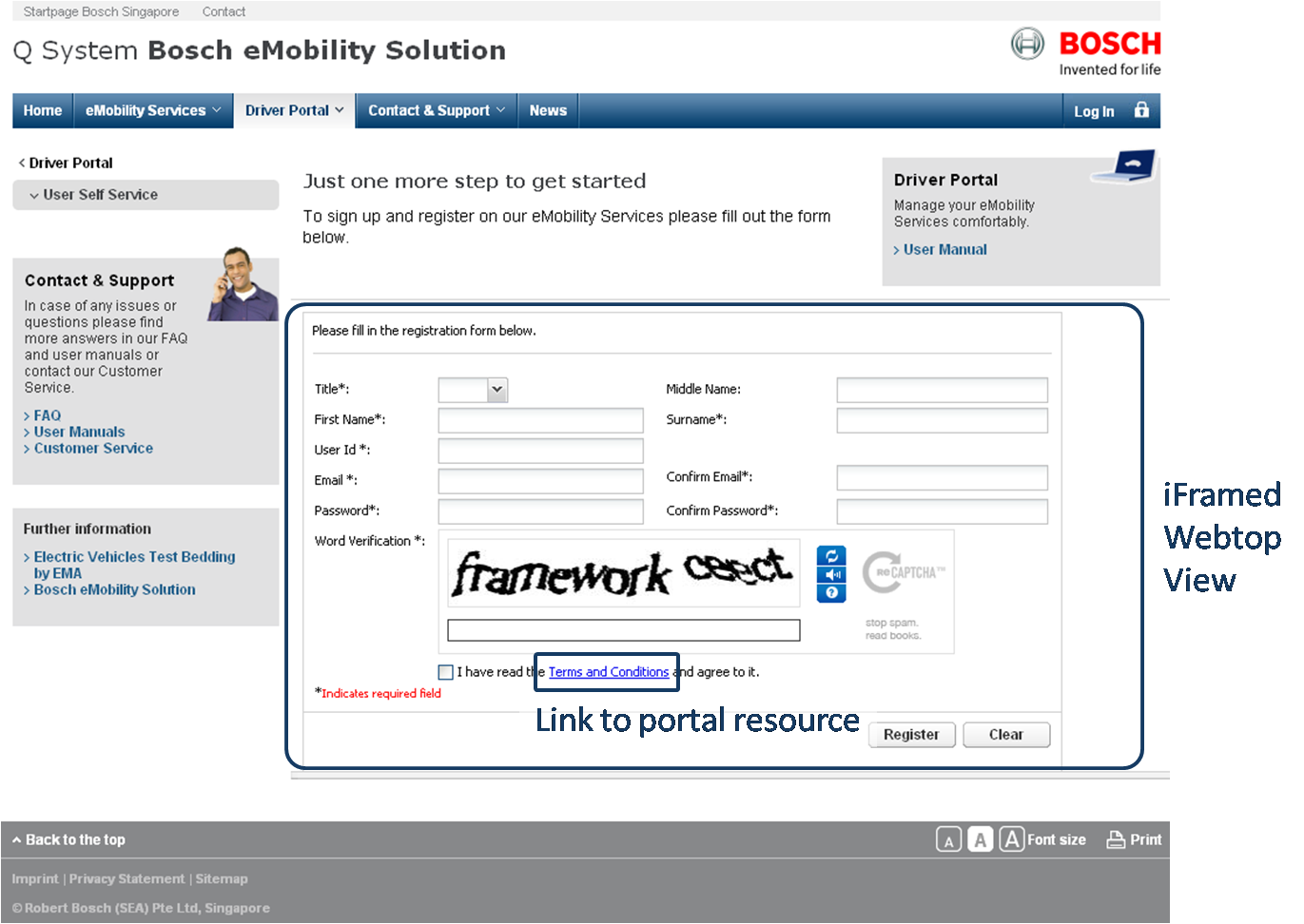


Figure : webTop - iFrame in a Portal

Regarding the difference **between web applications built with webTop and portal applications**, it has been stated that:

* A portal is a single point of entry to various web applications/services
  + Some D&D integration among different applications possible
* webTop is a mean to implement such web applications
  + It supports backend and frontend integration

In this perspective, webTop will typically be used **to create applications that can be exposed through portals** (e.g. as iFramed webTop views).

Additionally, designing standalone web applications – that can be optionally integrated in portals – comes with some advantages compared to designing web applications as portlets.

One of these advantages is the possibility to **delegate the execution of specialized applications logics on dedicated application servers**.

Following this pattern, a portal application server would then only support calling services and fetching UIs, and therefore **would not need to be scaled to support the integrated applications executions**.

## Web transfers performance

Network communications between the platform and its clients have been optimized through the use of **GZIP compression**. This functionality is applied to Direct Web Remoting calls as well for better service responsiveness.

In order to optimize bandwidth consumption, an option is also available to **aggregate and minify all Javascript and CSS resources** used by the platform Rich UI. This compression is applied **during the platform deployment phase**.

Last but not least, webTop supports the targeting of external resources, i.e. delivered by another domain. It can thus **integrate with a Content Delivery Network** for faster digital assets transfers.

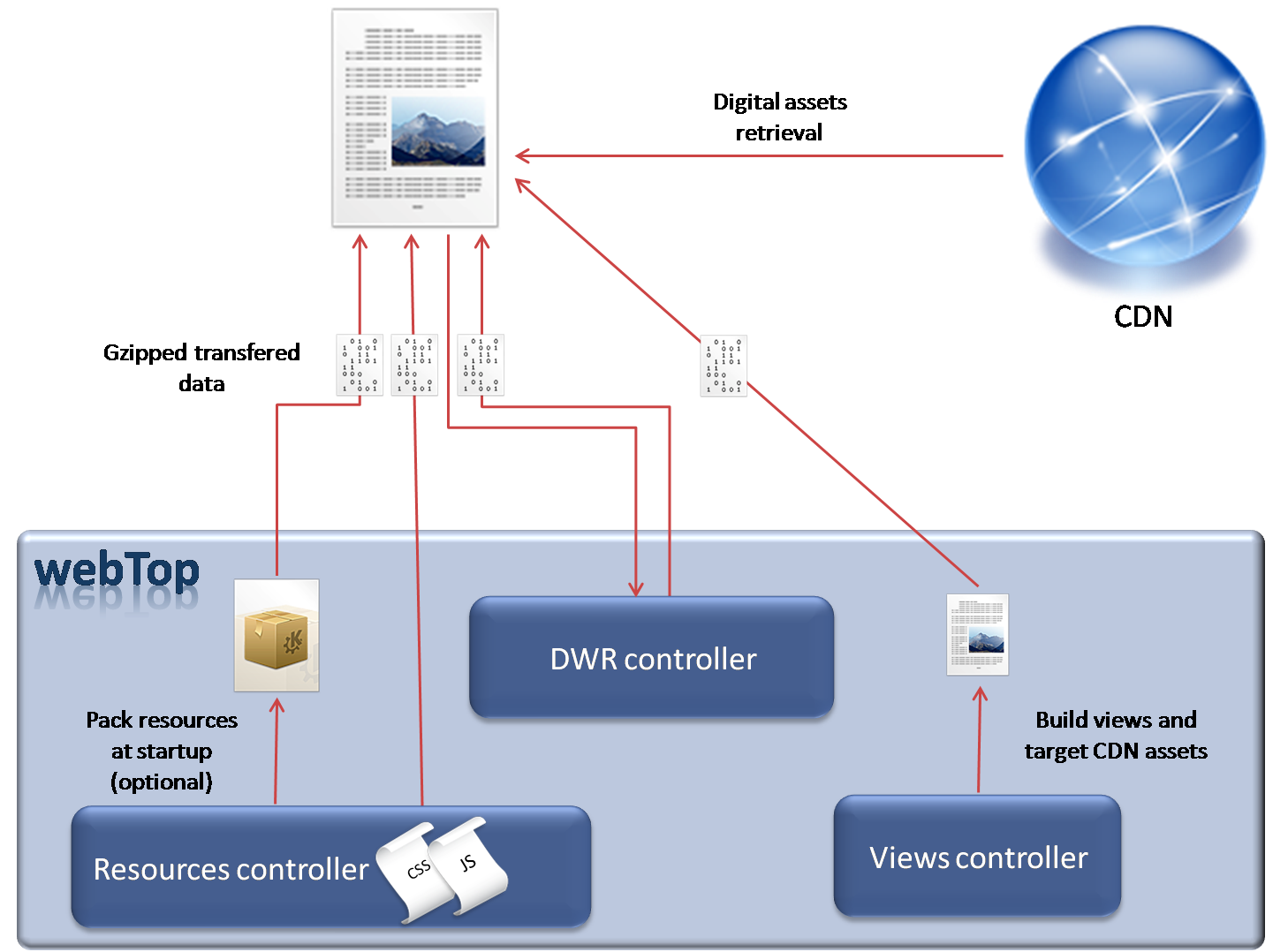


Figure : webTop Resources Handling

## Platform administration

webTop currently makes use of **JMX MBeans** to enable monitoring from **JConsole**. But it is up to the web application developer to write their own manageable beans in Java for custom administration of the platform.

However, **monitoring** of resources usage, number of platform users and number of queries within specific time frames is **under study for a future release**.

## Architecture

webTop is the base platform for delivering content and interacting with business services over the web. It thus implements the two following layers in a Web application environment:

* **Remoting Layer** by exposing/adapting business services to the Web,
* **Client Layer** by managing a customized rich client.



Figure : webTop System Positioning

As seen in the above diagram, webTop is the component **responsible for the aggregation of the business logic and the web-related aspects** to enable web remoting.

As for the internal webTop components enabling these features, they are **web-accessible controllers** as shown in the following diagram:

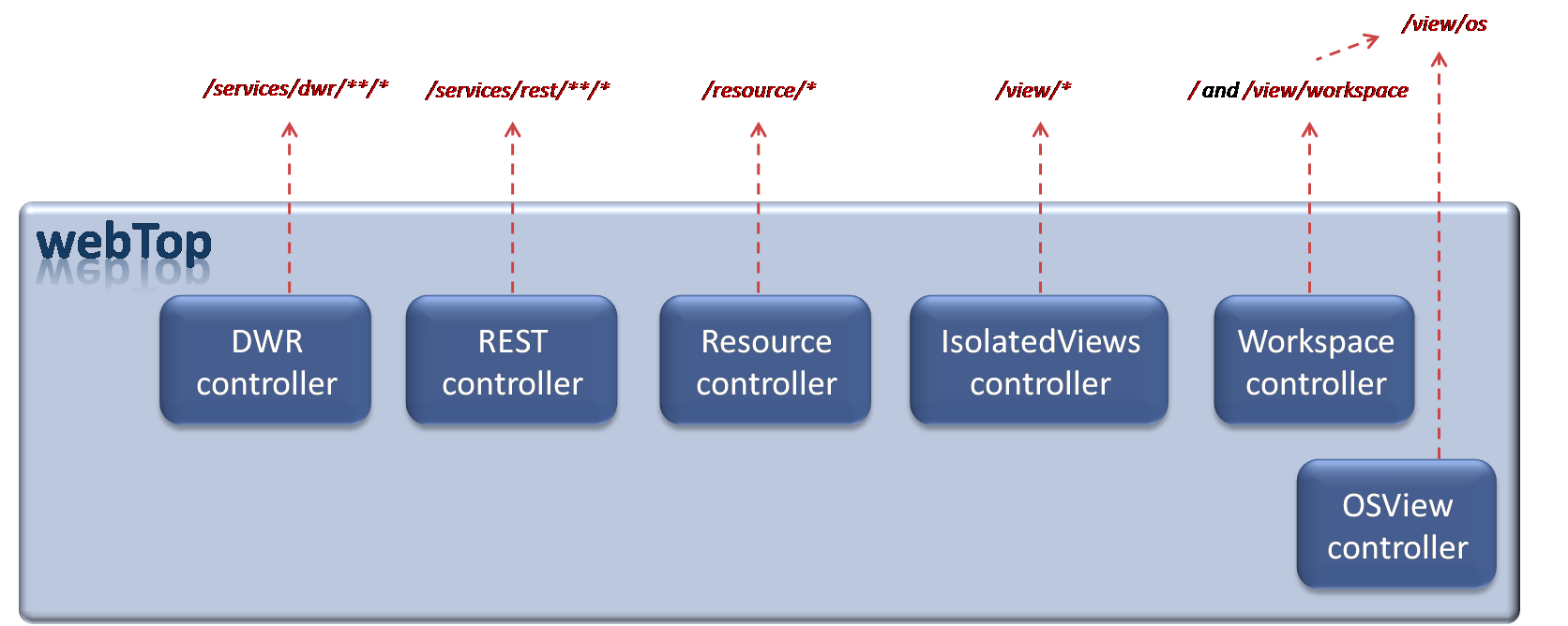


Figure : webTop URL-accessible Controllers

## Integration patterns

**webTop is available as a Java Library**, packaged in a JAR file. In order to run the webTop platform on an application server, **a simple web container (deployed WAR) is needed**. The web configuration (*web.xml*) should at least contain the following lines:

<listener>

<listener-class>com.innovations.webtop.web.context.WebtopContextListener

</listener-class>

</listener>

...

<servlet>

<servlet-name>*webtop*</servlet-name>

<servlet-class>com.innovations.webtop.web.context.WebtopDispatcherServlet

</servlet-class>

</servlet>

...

<servlet-mapping>

<servlet-name>*webtop*</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

To configure webTop itself, the application developer can then define the services and UI to expose in *webtop-app.xml*. The following is an example of such a configuration:

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<webtop>

<name>Demo</name>

<globalIncludes>

<css>classpath:demo.ux.roweditor/RowEditor.css</css>

<js>classpath:demo.ux.roweditor/RowEditor.js</js>

<css>classpath:demo.theme.gray/gray.css</css>

</globalIncludes>

<services>

**<!-- This service is available in the same web container (WAR). -->**

<dwr class=*"demo.web.LoginService"* scriptName=*"LoginService"* />

**<!-- This service is configured by Spring and can be either remote or local. -->**

<dwr type=*"spring"* scriptName=*"VRService"*>

<property key=*"beanName"* value=*"VRService"* />

</dwr>

</services>

<views>

<view id=*"demo.view.login"* src=*"demo.views/LoginForm.js"* jsClass=*"Demo.LoginForm"* alias=*"login"* />

<view id=*"demo.view.vr"* src=*"demo.views/VisualRulesClient.js"* jsClass=*"Demo.VisualRulesClient"* alias=*"vr"* />

</views>

</webtop>

The views defined above in the webTop application descriptors are **webTop views**. They can benefit from all webTop rich UI management features and be one of the following:

* **Custom view**
  + Exists only within the web container
  + Can use all the declared remote services of the platform
  + Can be a simple **iFrame wrapper to display external content**
* **Module view**
  + Imported from another **module JAR**
  + Packaging a feature or a whole application
  + Can be used for **mash-ups**

Currently, a module views and services can automatically be imported in webTop by specifying the module descriptor path in *webtop-app.xml*.

The idea behind modules is the following: “**write once, use in multiple applications**”.

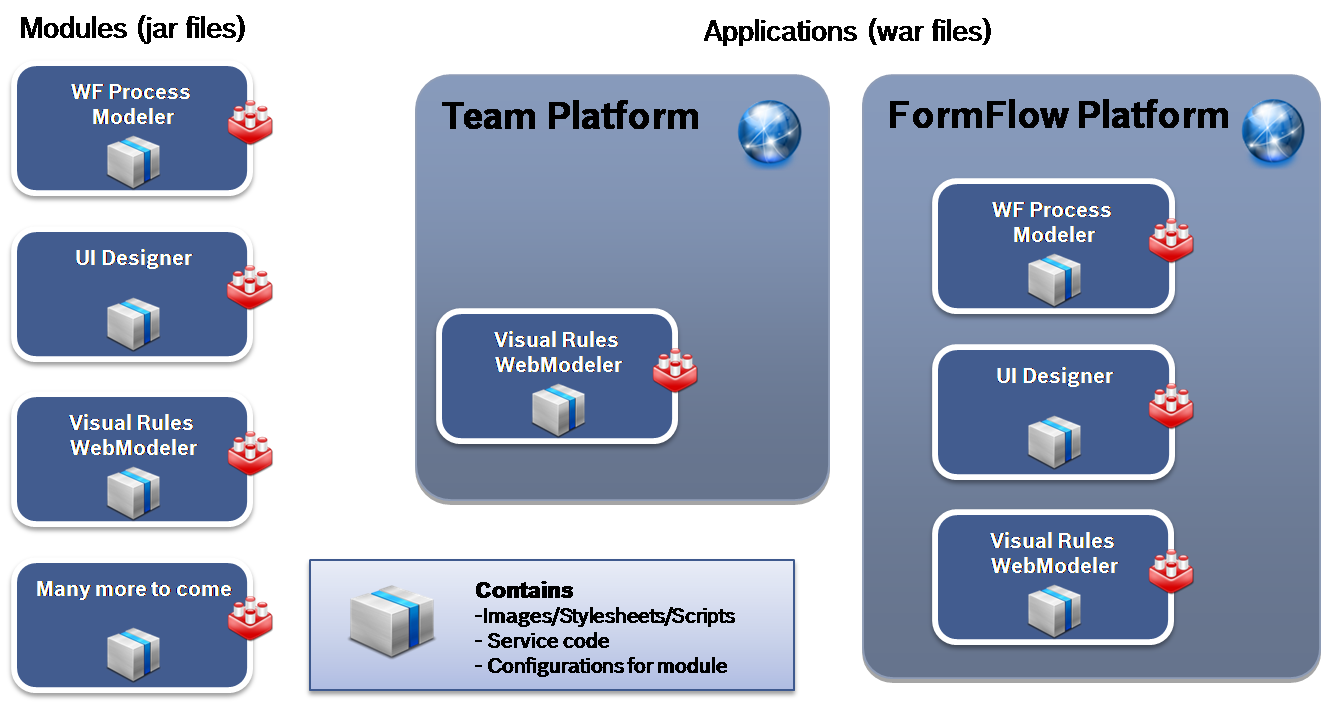


Figure : webTop Modules Sharing

The different possibilities to integrate external views and services make webTop extendable enough to be considered an **alternative to lightweight portal solutions**.

## Use cases

#### eMobility and WCMS integration

webTop has been successfully integrated in the eMobility architecture as it is the core component for the **Driver’s Portal**, the **User Self-Service** and the **Service Provider Portal**.

webTop mainly used features for these three solutions are:

* Portal integration (seamless integration with **WCMS** portal and **eNets payment system**)
* Security integration (**CAS** and **Https,** displaying views depending on user authorizations)
* Direct Web Remoting (with enabled gzipping)
* Desktop View (SPP), Isolated View (USS) and Traditional View (DP)
* Internationalized views

The following features repartition diagram also provides details about the integration pattern that has been used.



Figure : webTop use case – eMobility

#### FormFlow Platform

webTop is a core component of the FormFlow platform where it is **used as an integrated library** to power the FormFlow Platform UI and remote services.

webTop mainly used features within FormFlow are:

* Spring Security integration
* Desktop View
* Views display depending on user’s authorizations
* Direct Web Remoting (with enabled gzipping)
* Integration of generic contents (iFrames within views)
* Communication between views using the front-end messaging platform
* Module integration (WebViews, Workflow Modeler)

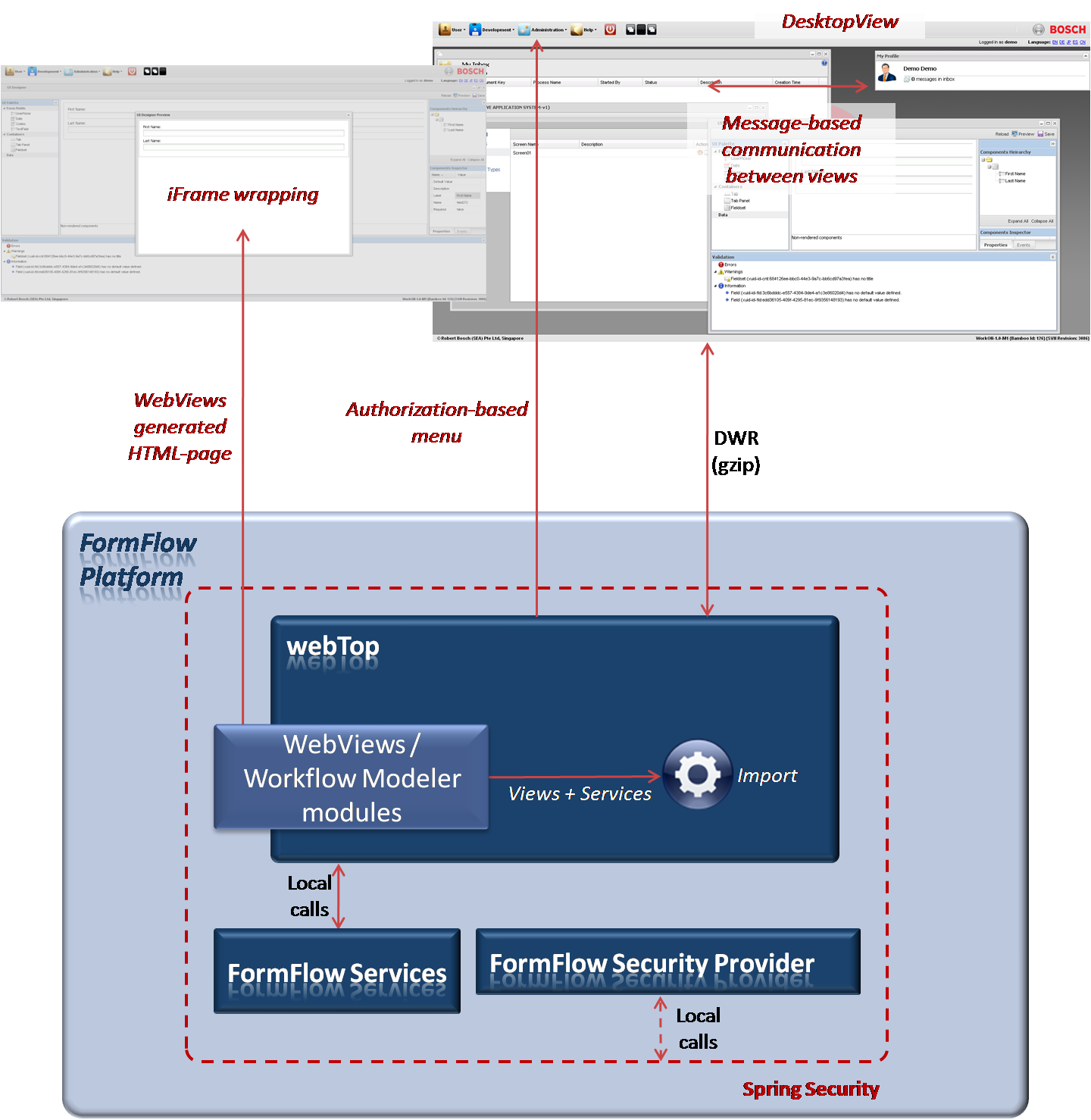


Figure : webTop use case - FormFlow Platform

# WebViews (This is still work in progress)

## Introduction

WebViews is a **dynamic web UI generator library**. Its core feature is to **generate rich interactive views based on descriptors** that can either be **retrieved from a repository** or **generated programmatically**.

WebViews also packages a **UI designer** (view + services) for end-users to model their own customized UIs and save the underlying descriptors to a repository. Besides, this UI designer is already provided as a **webTop module**.

## UI Designer

The UI designer is a rich web interface that allows user to build web pages managed by WebViews. Based on the **WYSIWYG** concept, it improves productivity when it comes to designing new views.

The UI itself exposes the following features in a nutshell:

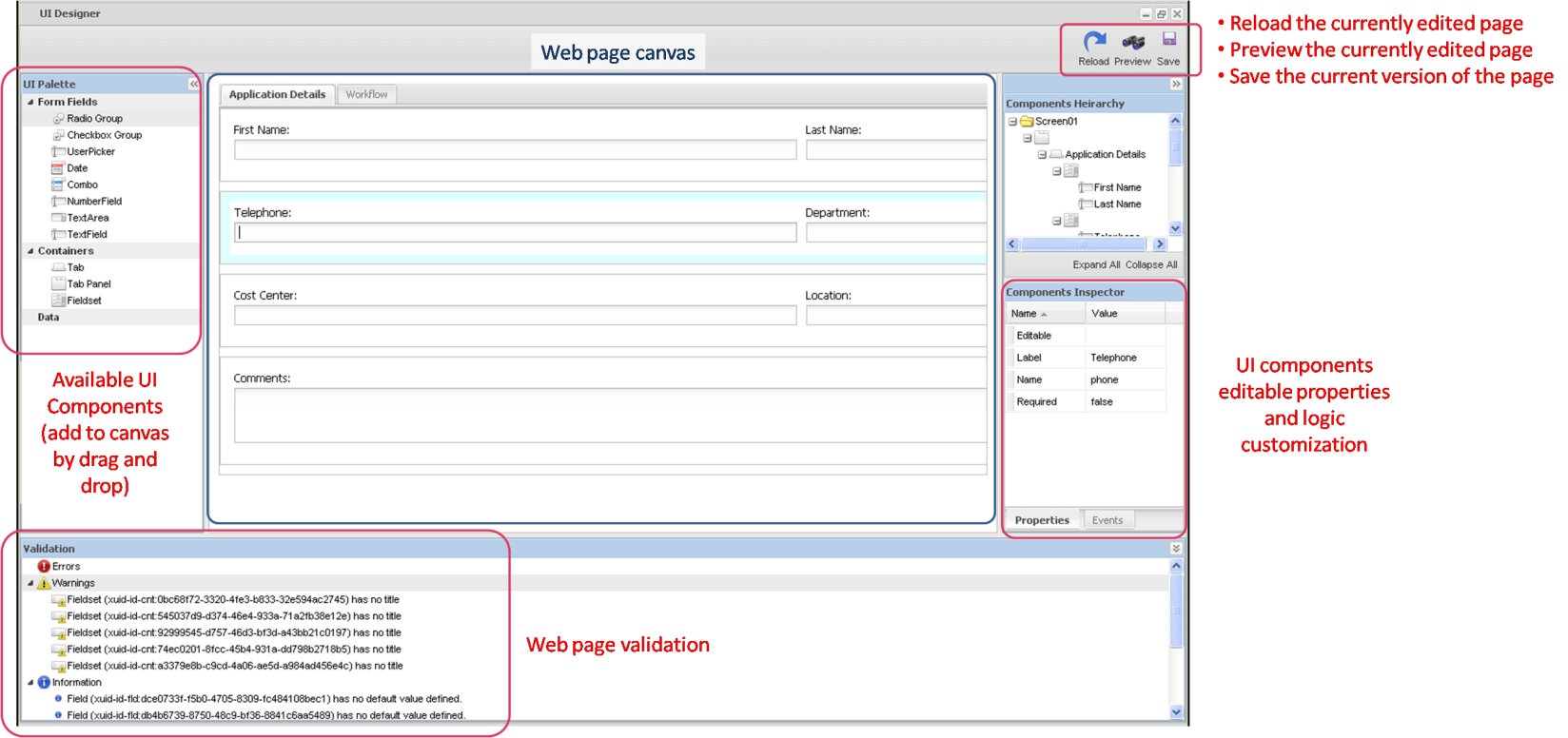


Figure : WebViews UI Designer Feature

More specifically:

* The “**Reload**” feature will replace the current version of the designed web page with the latest one available in the connected repository.
* The “**Save**” feature allows to commit the changes of the currently designed view to the connected repository.
* The “**Preview**” feature will query the View Processor/Renderer to display the current state of the edited web page.

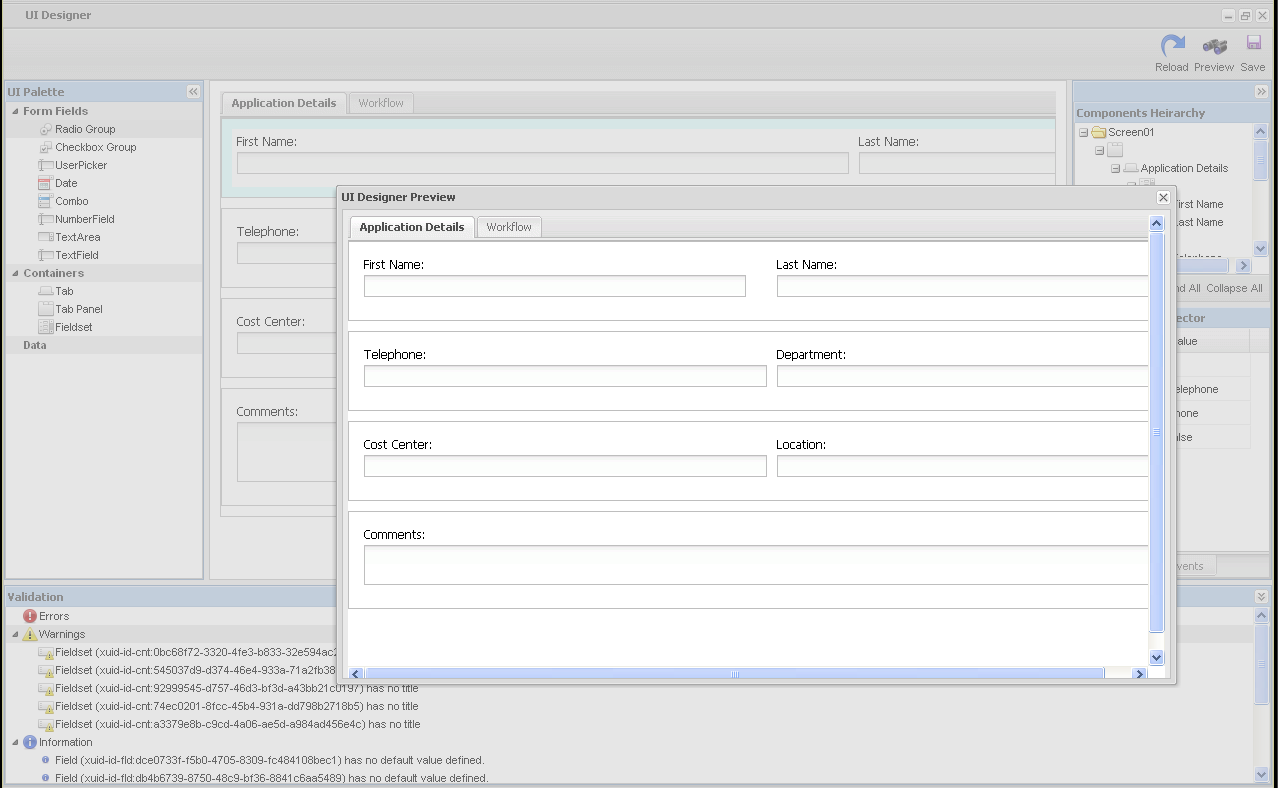


Figure : WebViews UI Designer Preview Feature

## Architecture

WORK IN PROGRESS

WebViews is a java library (JAR) and can be imported in any web application that needs dynamic UI generation.

Internally, it is wired as shown in the following diagram:

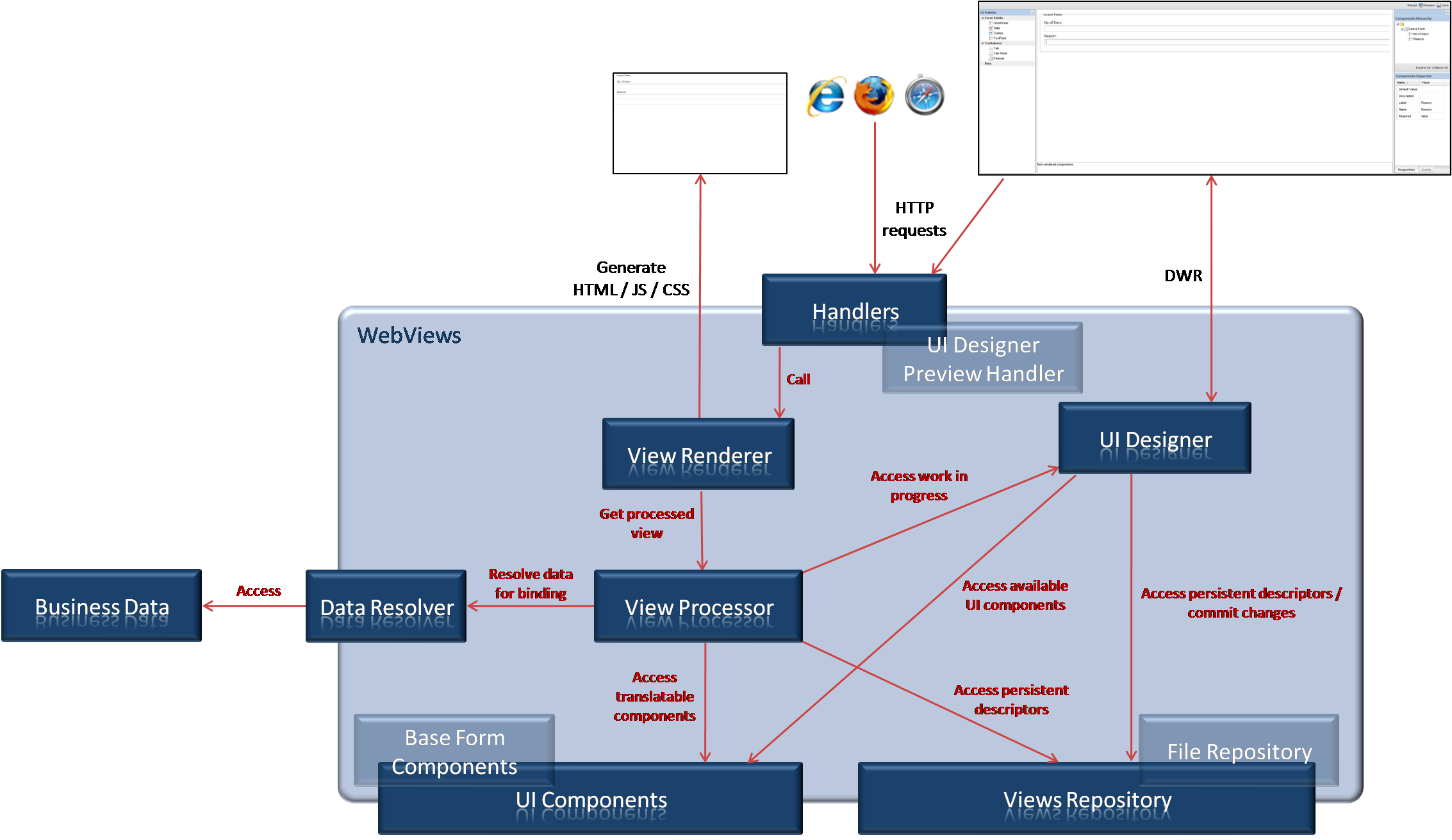


Figure : WebViews Architecture Overview

## Integration patterns

WORK IN PROGRESS

## Use cases

#### FormFlow Platform

WORK IN PROGRESS

WebViews is used by the FormFlow platform to provide user the ability to create, update and save forms. The way WebViews is integrated in the FormFlow Platform is described by the following diagram:

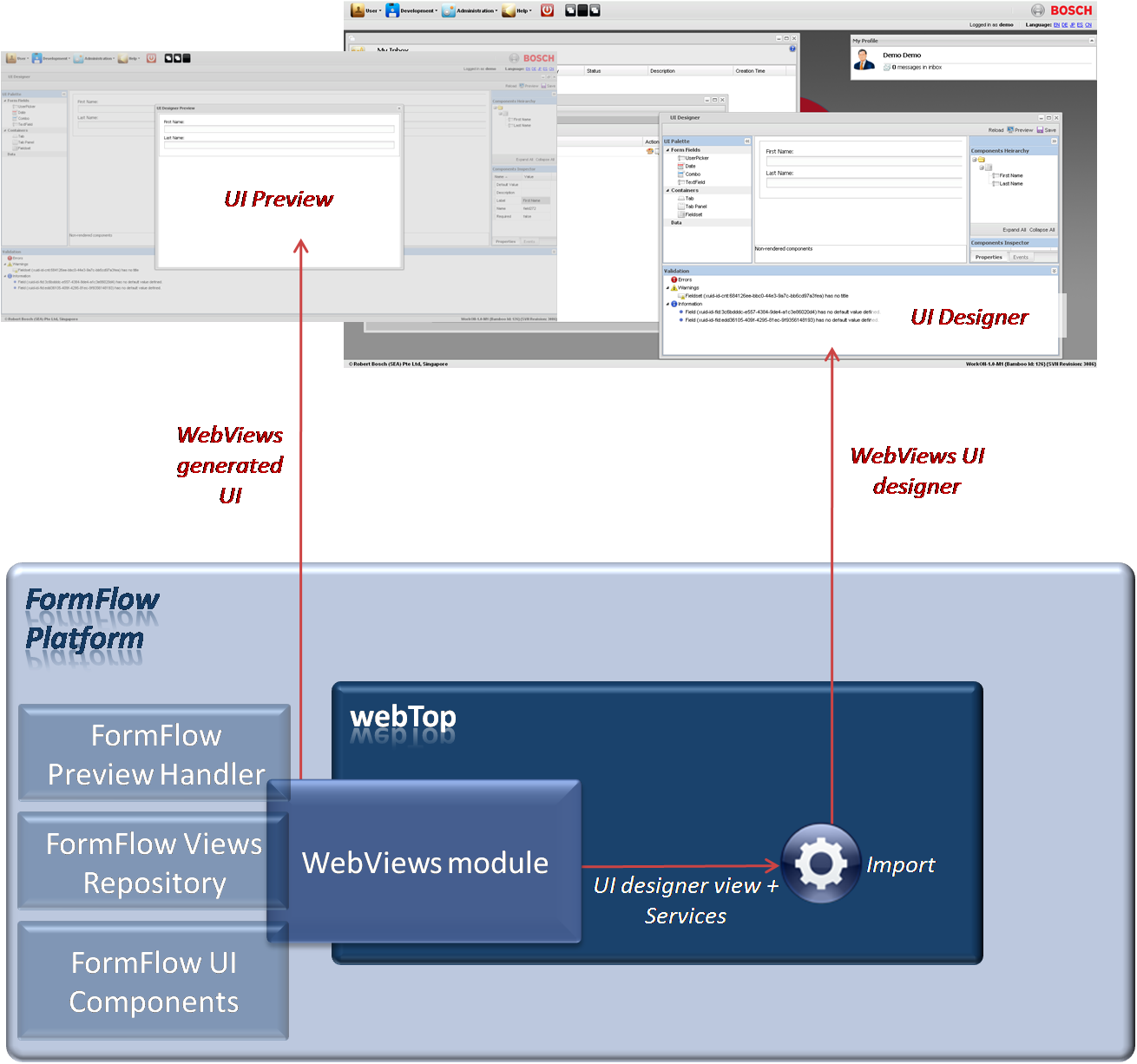


Figure : WebViews use case - FormFlow Platform

1. Multi-Service Framework [↑](#footnote-ref-2)