

OLAT Web GUI Framework

Summary and technical overview

Version: 1.2
Date: October 2006
Author: Mike Stock, Felix Jost

Table of Contents

1 Management summary.....	3
2 Concepts in detail.....	4
2.1 Overview.....	4
2.2 Programming concepts.....	4
2.2.1 Swing-like programming.....	4
2.2.2 Flexible Layout.....	4
2.2.3 Component based.....	5
2.2.4 Event dispatching.....	5
2.2.5 Separation of logic and layout.....	6
2.2.6 Exception handling.....	7
2.2.7 Summary of programming concepts.....	7
2.3 Ease of developing and debugging.....	8
2.3.1 Focus on concern.....	8
2.3.2 Visual development / debugging assistance.....	9
2.3.3 Passing through of layout and i18n changes.....	10
2.3.4 Online translation tool.....	10
3 Advanced Features.....	12
3.1 Graceful degradation.....	12
3.2 AJAX / Web 2.0 -Mode.....	12
3.2.1 Better performance.....	12
3.2.2 Improved user experience.....	12
4 Architectural reference.....	13

1 Management summary

The OLAT Web GUI Framework is a 100% pure Java Framework which speeds up development, testing, debugging, and maintenance of web applications.

Some of its unique features are:

- Very similar to Sun's Java Server Faces (JSF), but much more powerful.
- Focuses not only on the reuse of components, but also on the reuse of GUI flows / Business controllers.
- Dual mode (configurable per user session) to support legacy clients:
 - Bare bone HTML: No Javascript or advanced DOM tree modifications required. Workflows are guaranteed to run, but some convenience functions will not be available.
 - AJAX mode: Relies on Javascript and DOM replacement. This provides convenience functions such as cursor positioning and form modification control. Furthermore, DOM replacement considerably reduces traffic and client render time resulting in a noticeably faster GUI and in improved user experience.
- Active development by the University of Zurich, JGS goodsolutions GmbH (www.goodsolutions.ch), BPS System, Germany (<http://www.bps-system.de/>), and by the Open Source community.
- Debug/development mode which integrates additional debugging/development information and associated tools directly into the GUI of your web application.
- Proven stability, even under heavy load. In production for over three years on our reference system at the University of Zurich.
- Open Source and free of charge, licensed under the flexible Apache BSD Open Source Licence which allows for closed source modifications.
- Combines efficient Swing-styled programming with the powerful control of layout and look-and-feel by means of HTML fragments and CSS.
- Leverages on other well established Open Source software.

For more detailed documentation please refer to the Open Source Community site at <http://www.olat.org>

2 Concepts in detail

2.1 Overview

The GUI framework is component-based, follows the stateful MVC (Model-View-Controller) paradigm, and focuses on the reuse of components and, most important, of GUI flows (also called Business controllers).

It is similar to Java Server Faces, but more powerful. Programming the GUI is very similar to developing with Swing and its Swing Windowing Toolkit (SWT). The framework provides developers with standard components such as panels, containers with layouting, forms, tables and alike. Developers familiar with rich client programming with Swing should be able to quickly get acquainted with the framework.

Gui workflows are grouped into reusable controller classes, so that e.g. a user search (consisting of a search form, followed by a list of found matches, followed by a selection of a user) can be reused from within many other controllers which need a user search GUI workflow.

The framework can support any language since it is based on the UTF-8 character encoding for storing both localization and user data. There is an online translations tool provided by the framework which facilitate the process of translation.

2.2 Programming concepts

2.2.1 Swing-like programming

Similar to Swing, components are stateful, so that a programmer does not need to worry about restoring the state of a workflow.

The Business Controllers listen to events of associated components and then react to it by calling managers for the business logic and by updating the render tree (e.g. replace a search form with the search result table).

The view of an application is conceptually broken down into the following components:

- A `Window` component is the top-level component and represents a browser window.
- Any Business Controller has its specific stateful subcomponents. All components are linked together in a render tree. The root of this tree is attached to a aforementioned Window component.
- `Containers` are components responsible to layout their children, and thus, recursively, generate an HTML representation of their childrens' current state.

2.2.2 Flexible Layout

The `VelocityContainer` is one concrete implementation of a Container which is able to render Velocity templates. The Velocity Templating Engine is an Apache Open Source project and provides a very fast and reliable Java-based template engine. It permits anyone to use a simple yet powerful template language to reference objects defined in Java code much as is the case with JSP.

The combination of HTML-Fragments and together with Cascading Stylesheets (CSS) give you powerful control of the overall graphical design. Common layouting of containers like Tabbed Panes, Stackable Panels and alike are also provided by the framework. Furthermore, Containers for any other presentation engine such as JSP, JavaServer Faces can be easily integrated into the

framework.

2.2.3 Component based

A `Component` is a visual representation of a `Form`, a `Table` and so on. One or more `Controller` can listen to events which are fired when the user e.g. submits a form or clicks a link. Each component must provide its `HTML-Renderer` (method `getHTMLRendererSingleton` (interface `ComponentRenderer`) which knows how to render its state in HTML.

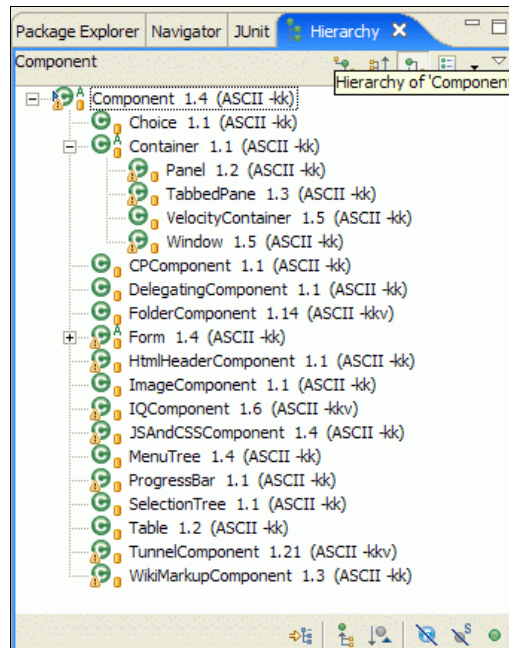


Illustration 1: Component hierarchy

2.2.4 Event dispatching

The GUI Framework has mainly two things to do upon each browser-click from a user:

1. Dispatch the request to the component which was clicked by the user in the browser
 - Find the component (with the component-id of the url) in the component tree of the content pane of the window.
 - Call `dispatchRequest(UserRequest ureq)` of the component, so that it can update its state.
 - The component will fire an event to the listening controllers if appropriate (e.g. "Treenode clicked" with the id of the clicked node or "Form submitted").
 - The controller(s) which receive the event
 - update their internal state, which means most of the times to call business logic and
 - as a result, the controller updates some of the components it owns (e.g. advance in a wizard step) and
 - might fire an event to listening "parent" controllers. (e.g. an `UserChosenEvent` when a user was chosen.

2. Prepare a response to the client by either

2.1. Render the new state in HTML.

The rendering is done by collecting the render output of each individual component registered in the render tree. It is only after the complete render tree has been traversed that a response is sent to the client's webbrowser. This has the advantage that a user never sees those half-baked screens in case of an error during the render process. The resulting page will either be sent as a whole or a decent error screen is produced with a reference number for support staff to further track down errors.

2.2. Deliver a resource such as a PDF-Document or any data that can not be rendered in HTML

To avoid malformed URIs, the concept of an `URLBuilder` class has been introduced. This class is responsible for building all links for the whole application and encodes necessary parameters which are internal to the framework such as window id, component id, timestamp, ajax mode, etc. transparently into the final URI.

```

/**
 * @see org.olat.core.gui.control.DefaultController#event(org.olat.core.gui.UserRequest, org.olat.core.gui
 */
public void event(UserRequest ureq, Component source, Event event) {
    if (source == groupmemberview) {
        if (event.getCommand().equals("adduser")) {
            userSearchController = new UserSearchController(ureq, getWindowControl(), true);
            userSearchController.addControllerListener(this);
            getWindowControl().pushToMainArea(userSearchController.getInitialComponent());
        }
    }
}

/**
 * @see org.olat.core.gui.control.DefaultController#event(org.olat.core.gui.UserRequest, org.olat.core.gui
 */
public void event(UserRequest ureq, Controller sourceController, Event event) {
    if (sourceController == userSearchController) {
        if (event instanceof UserChosenEvent) {
            Identity ident = ((UserChosenEvent)event).getChosenIdentity();
            // start using the chosen identity
        }
        // else cancelled
        getWindowControl().pop();
    }
}

```

Illustration 2: Observer pattern event listeners

2.2.5 Separation of logic and layout

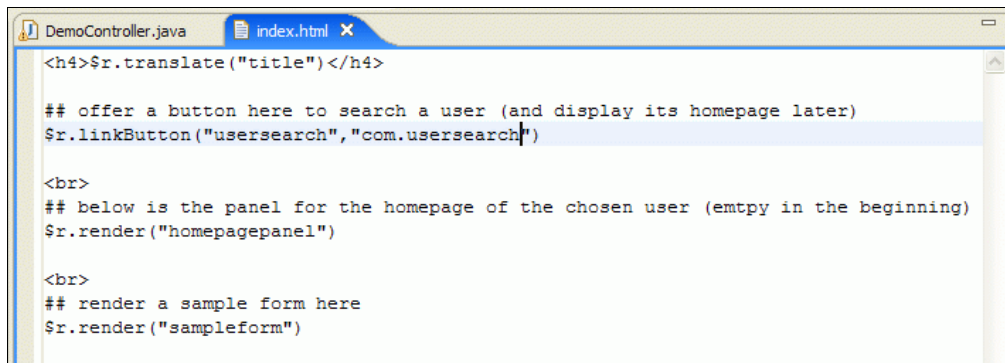
All GUI flows (Business logic as seen on the screen) are encapsulated in reusable `Controller` classes (and the `Manager` classes they use to execute the Business logic), whereas the layout can be controlled by mainly modifying CSS and writing HTML-Fragments.

- CSS is thoroughly used in the framework so that almost all layout customization can be done with it (even to the extent of defining images and icons)
- HTML fragments are used for the main layouting of each controller. All those HTML Fragments are later recursively composed to a full HTML page which is sent to the user.

The example below should give you an idea of how HTML fragments might look like.

- The first paragraph defines a button wich links to a workflow to search for a user.
- The second paragraph defines a yet empty panel which acts as a placeholder. Once the user search workflow triggered by the button defined in the first paragraph finishes, the resulting user's homepage will be shown here.

- Finally, the third paragraph of this example HTML fragment will render a sample form.

A screenshot of a code editor window with two tabs: 'DemoController.java' and 'index.html'. The 'index.html' tab is active, showing the following code:

```
<h4>${r.translate("title")}</h4>

## offer a button here to search a user (and display its homepage later)
${r.linkButton("usersearch","com.usersearch")}

<br>
## below is the panel for the homepage of the chosen user (empty in the beginning)
${r.render("homepagepanel")}

<br>
## render a sample form here
${r.render("sampleform")}
```

Illustration 3: HTML fragments for presentation

2.2.6 Exception handling

The framework is based on the “*unchecked exception*” principle. This means that the developer does not need to care about exceptions in general. Exceptions travel up the stacktrace and are caught on top level to produce a decent “*red screen*” with information on date and time the error occurred and an error number which enables support personnel to track the error if needed. A convenient web interface is provided for the support personnel to search the error logs based on the information the user is given on the screen in case of an unexpected error.

Detailed logs also provide full Java stacktraces and additional framework related information such as which component of which controller caused or triggered the error.

2.2.7 Summary of programming concepts

The following diagram illustrates all of the aforementioned concepts with a simple user search workflow serving as an example.

You'll find the relationship between controllers and components, an outline of their event model and their view.

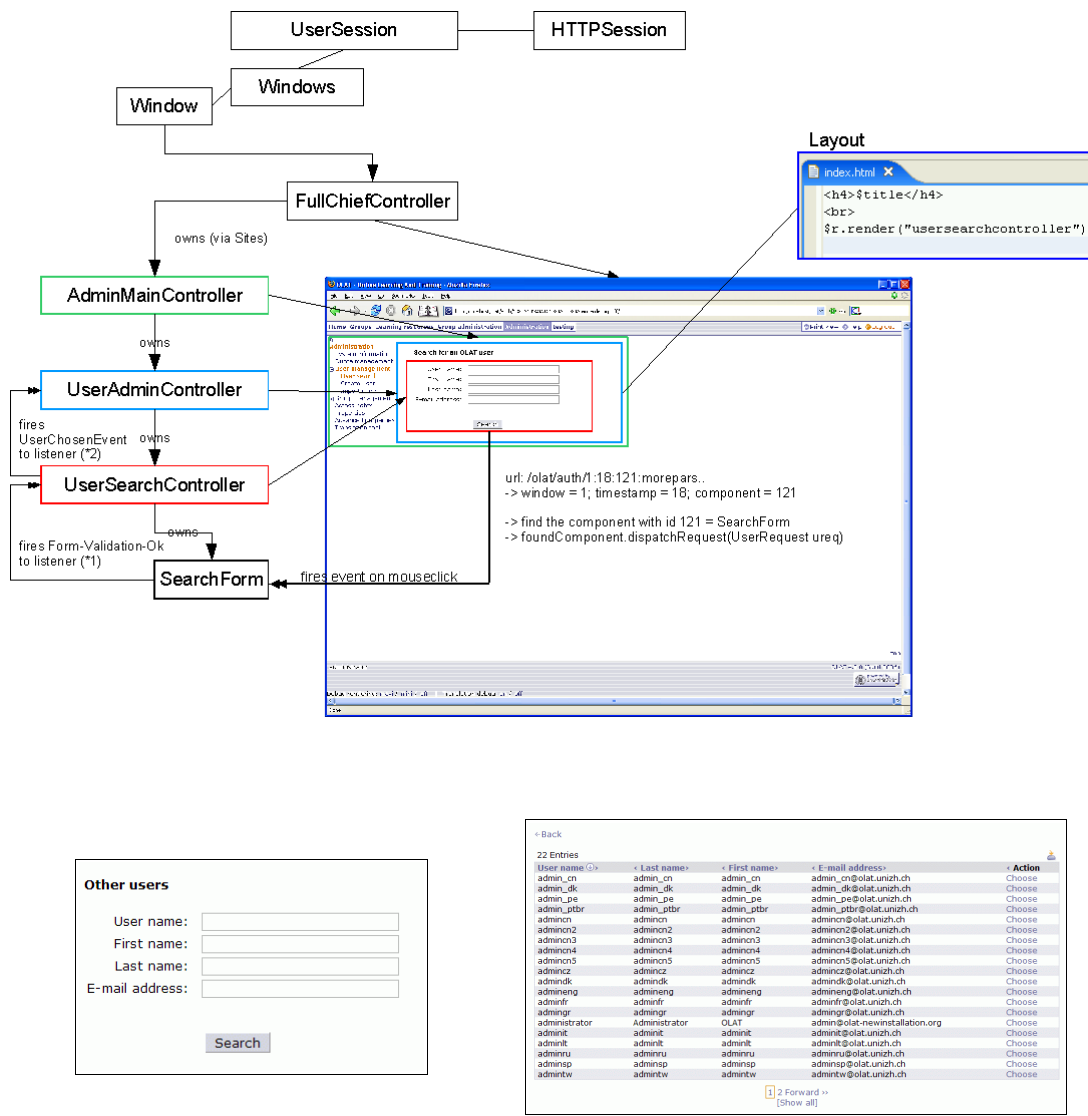


Illustration 4: Illustration of a "User search" workflow

2.3 Ease of developing and debugging

The framework supports the developer with some unique built-in features. All these features are instantly available through a single switch in the framework configuration (Debug Mode) without the need of modifying your code. Once your application is ready for production, simply turn off Debug Mode and your set to deploy.

2.3.1 Focus on concern

All code and data such as graphics, HTML fragments and translation are grouped together in one place. From a developers point of view, this eases the process of development by having all related files in one place. From a deployment point of view, it enables new functionality to be deployed through simply packaging the structure into a single jar file.

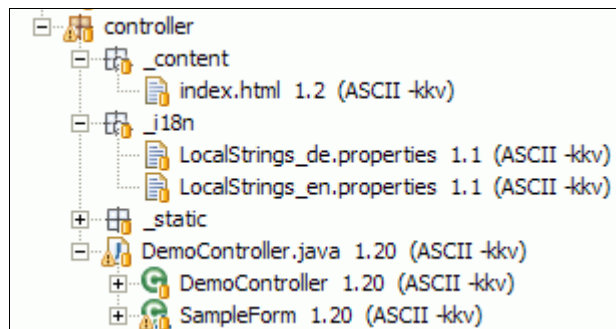


Illustration 5: Code, *i18n* and content all in one place

2.3.2 Visual development / debugging assistance

In Debug Mode, the framework implants additional debugging information directly into the GUI of your application. This allows easier identification of use of components, their translators and associated Controllers and their associated listeners. Links to tools such as HTML fragments editors and the translation tools are provided where applicable.

The below screenshot shows the OLAT e-Learning Application which is based on the framework in full Debug Mode.

Illustration 6: GUI debug stack

2.3.3 Passing through of layout and i18n changes

The Debug Mode helps in understanding the GUI stack structure. In complex applications, it is sometimes easier to find out how the View of your application is composed by visually identifying each component directly in the GUI, rather than analyzing the actual code. Refreshing HTML fragments and localized string properties enable the developer to change or fix broken layouts and translation keys on the fly. The yellow highlighted component, visible in Illustration 5, produces a floating box with additional information:

- *type*, the java class name of the component.
- *listeners*, a list of event listeners attached.
- *cListeners*, a list of controller event listeners.
- *translator*, shows where the keys come from and in which language. Clicking on the link brings up a new browser window with the possibility to interactively fix translation keys.
- *nesting level*, a counter for how deep in the render tree this component is located.

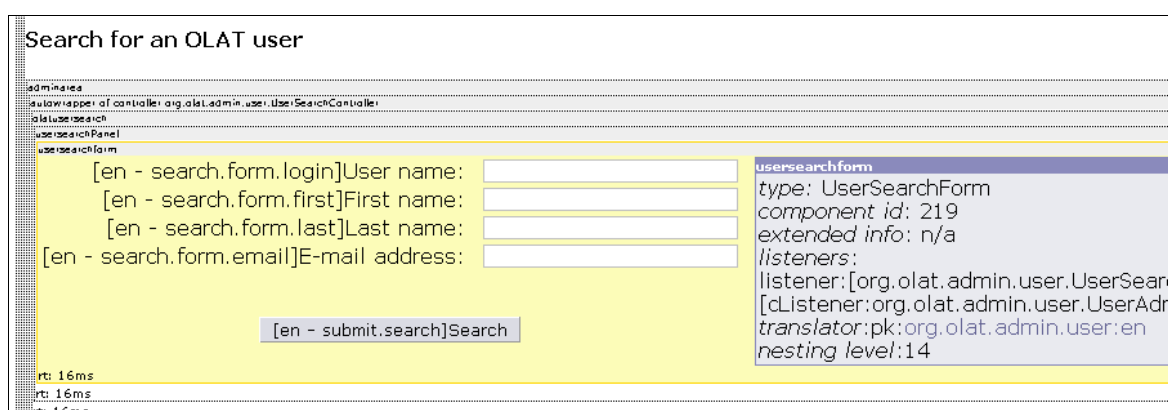


Illustration 7: GUI debug listeners box

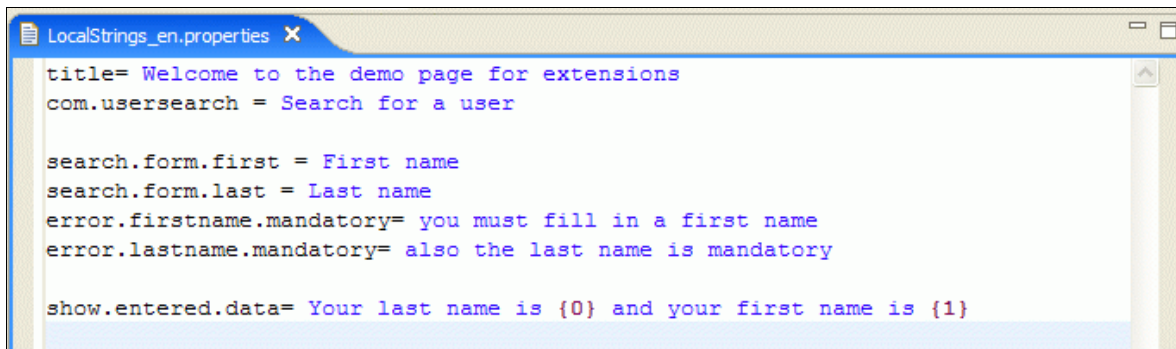
2.3.4 Online translation tool

All textual information within the framework or your application can be internationalized. Internationalized versions of textual information is held in a simple property file, one for every language the application has been translated into.

There is no need for the translating staff to work on the code directly. The framework provides a much easier way through its integrated online translation tool.

At the time of this writing, the framework supports the following languages:

English, Deutsch, Français, Español, Zhongwen, Cestina, Dansk, Hellinika, Italian, Persian, Polski, Portugues, Russkij.



```

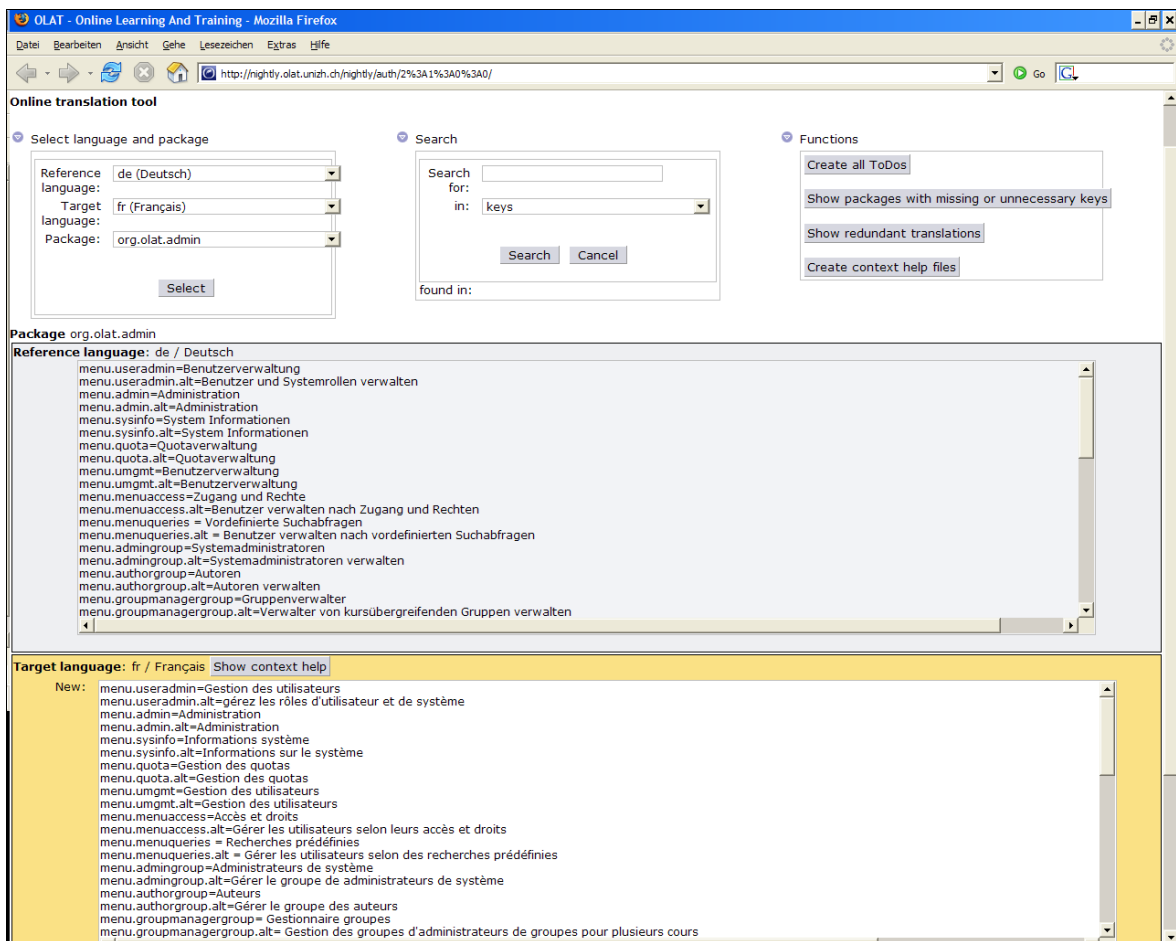
title= Welcome to the demo page for extensions
com.usersearch = Search for a user

search.form.first = First name
search.form.last = Last name
error.firstname.mandatory= you must fill in a first name
error.lastname.mandatory= also the last name is mandatory

show.entered.data= Your last name is {0} and your first name is {1}

```

Illustration 8: Full support for internationalisation (i18n) through simple property files



OLAT - Online Learning And Training - Mozilla Firefox

http://nightly.olat.unizh.ch/nightly/auth/2%3A1%3A0%3A0/

Online translation tool

Select language and package

Reference language: de (Deutsch)
 Target language: fr (Français)
 Package: org.olat.admin

Select

Search

Search for:
 in: keys

Search Cancel

found in:

Functions

Create all Todos
 Show packages with missing or unnecessary keys
 Show redundant translations
 Create context help files

Package org.olat.admin

Reference language: de / Deutsch

menu.useradmin=Benutzerverwaltung
 menu.useradmin.alt=Benutzer und Systemrollen verwalten
 menu.admin=Administration
 menu.admin.alt=Administration
 menu.sysinfo=System Informationen
 menu.sysinfo.alt=System Informationen
 menu.quota=Quotaverwaltung
 menu.quota.alt=Quotaverwaltung
 menu.umgmt=Benutzerverwaltung
 menu.umgmt.alt=Benutzerverwaltung
 menu.menuaccess=Zugang und Rechte
 menu.menuaccess.alt=Benutzer verwalten nach Zugang und Rechten
 menu.menuqueries = Vordefinierte Suchabfragen
 menu.menuqueries.alt = Benutzer verwalten nach vordefinierten Suchabfragen
 menu.admingroup=Systemadministratoren verwalten
 menu.admingroup.alt=Systemadministratoren verwalten
 menu.authorgroup=Autoren
 menu.authorgroup.alt=Autoren verwalten
 menu.groupmanagergroup=Gruppenverwalter
 menu.groupmanagergroup.alt=Verwalter von kursübergreifenden Gruppen verwalten

Target language: fr / Français Show context help

New:

menu.useradmin=Gestion des utilisateurs
 menu.useradmin.alt=gérez les rôles d'utilisateur et de système
 menu.admin=Administration
 menu.admin.alt=Administration
 menu.sysinfo=Informations système
 menu.sysinfo.alt=Informations sur le système
 menu.quota=Gestion des quotas
 menu.quota.alt=Gestion des quotas
 menu.umgmt=Gestion des utilisateurs
 menu.umgmt.alt=Gestion des utilisateurs
 menu.menuaccess=Accès et droits
 menu.menuaccess.alt=Gérer les utilisateurs selon leurs accès et droits
 menu.menuqueries = Recherches prédéfinies
 menu.menuqueries.alt = Gérer les utilisateurs selon des recherches prédéfinies
 menu.admingroup=Administrateurs de système
 menu.admingroup.alt=Gérer le groupe de administrateurs de système
 menu.authorgroup=Auteurs
 menu.authorgroup.alt=Gérer le groupe des auteurs
 menu.groupmanagergroup= Gestionnaire groupes
 menu.groupmanagergroup.alt= Gestion des groupes d'administrateurs de groupes pour plusieurs cours

Illustration 9: Online translation tool for comfortable translation

3 Advanced Features

3.1 Graceful degradation

The framework uses Javascript for some of its comfort functionality. On browsers that do not support Javascript or if you decide to disable Javascript for security reasons, the framework gracefully degrades on its technical features without compromising the business functionality of your web application.

The minimum requirements for a web browser are:

- HTML 4.01 transitional capable (e.g. Internet Explorer 5.x, Firefox 1.x, Safari)
- Standard J2EE session cookie (does not store any personal data on the client)

3.2 AJAX / Web 2.0 -Mode

The benefits of having AJAX in your web application are mainly twofold:

- better performance and less bandwidth requirements (through means of generic DOM replacement)
- improved user experience (auto completion, drag and drop, push)

3.2.1 Better performance

The following figures we've gathered from the OLAT e-Learning Application at the University of Zurich with an average of 200-500 concurrent users, should give you a rough idea of what a difference AJAX makes with regard to overall performance.

	<i>Render time on server</i>	<i>Total data to be transferred</i>	<i>Render time on client's browser</i>
Standard mode	7 ms	20 kb	0.8 s
AJAX mode	3 ms	5 kb	0.4 s

Our framework uses the AJAX-DOM-Replacement-Technology in a very safe and reliable way:

- AJAX can be turned on and off either globally, per browser, or per user!
- Developer's code does not depend on the mode
- No asynchronous code needed (no XMLHttpRequest usage)

3.2.2 Improved user experience

If AJAX is enabled, the framework uses server-side technology together with well established client-side AJAX libraries to deliver an improved user experience.

With AJAX, we're able to provide

- Input field auto completion (suggest as you type)
- Drag and drop
- Push technology

We strongly encourage application developers to offer such functionality as a bonus only. All business functionality must be achievable also when Javascript is turned off.

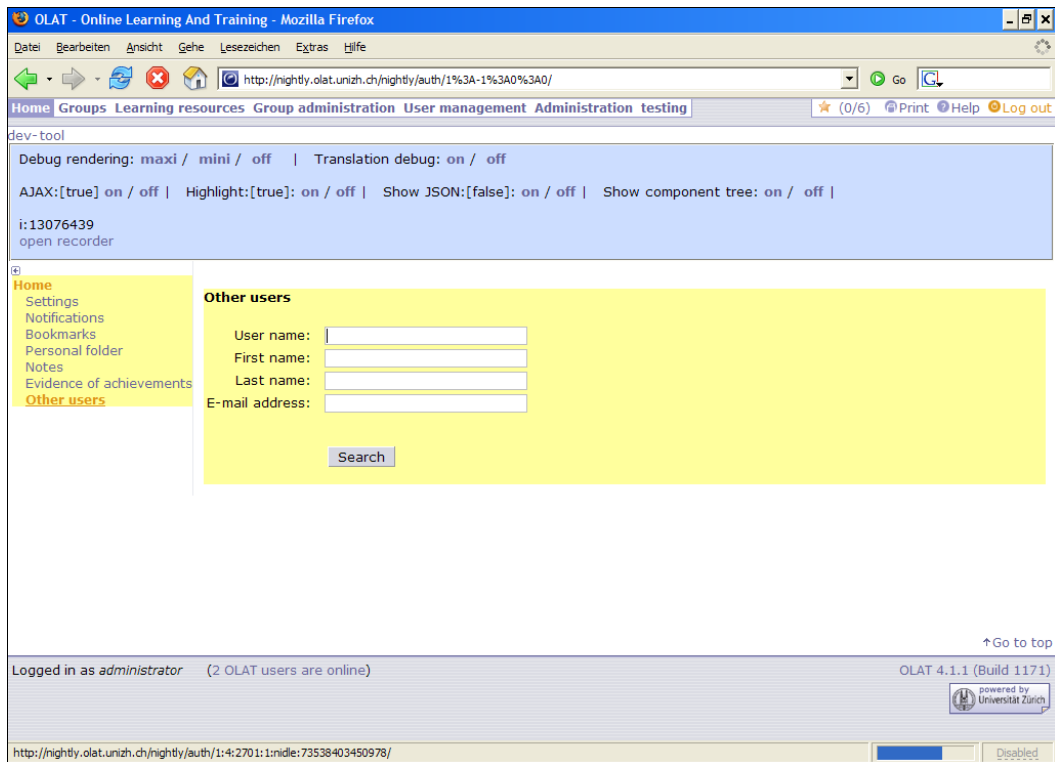


Illustration 10: DOM replacement (AJAX mode) in action (yellow background)

To help developers understand how AJAX affects their application, the framework provides a highlighting mechanism in Debug Mode which visually shows exactly what part(s) of the application's GUI are affected and will be replaced.

4 Architectural reference

This last illustration gives a quick overview of some of the frameworks architectural designs.

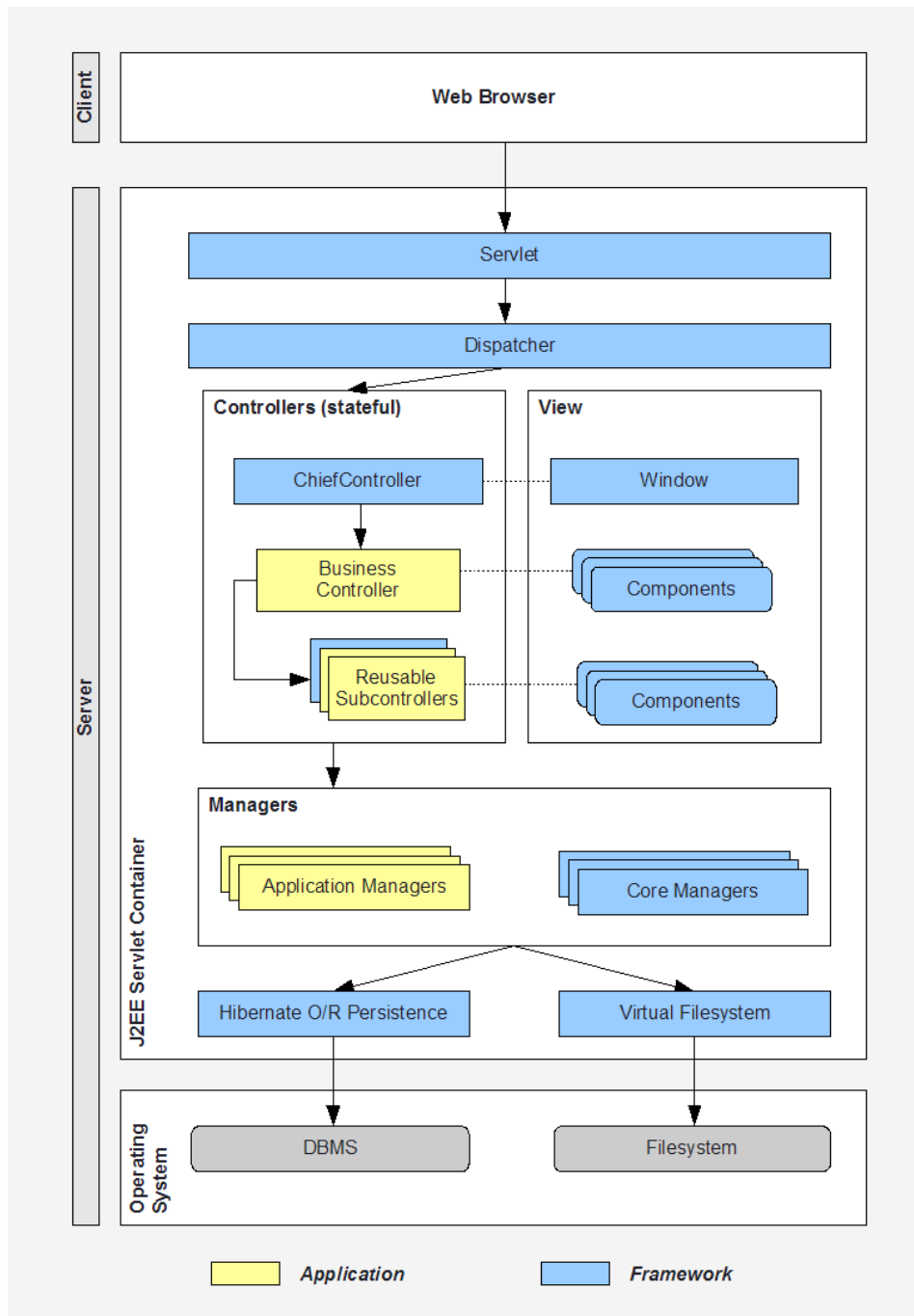


Illustration 11: Architectural layers