LIFERAY AND ALFRESCO: A CASE STUDY IN INTEGRATED ENTERPRISE PORTALS

Milorad Filipović*, Gajo Petrović*, Aleksandar Nikolić*, Vidan Marković**, Branko Milosavljević*
Faculty of Technical Sciences, Novi Sad, Serbia
****DDOR, Novi Sad, Serbia***
{mfili, gajop, anikolic, gladic, mbranko}@uns.ac.rs, vidan.markovic@ddor.co.rs

Abstract – *In this paper we propose a solution for the development of enterprise portal-like web applications that is based on the Liferay portal and the Alfresco document repository. Combining a robust and feature-rich platform for building the application layer such as Liferay with the industry standard document management system such as Alfresco enables developers to rapidly provide their customers with a custom portal web application that suit their needs. Emphasis of this paper is on extending the combined solution by implementing a custom activity list that shows integrated user activities from both Liferay and Alfresco, a feature missing in the original combined platform.*

Keywords – *Liferay, portal, Alfresco, document management, CMS, system integration*

1. INTRODUCTION

Building an enterprise information system is always a challenging task. Even though great effort has been invested in business application standardization and automation of the development process [1], each project and each customer bring their own custom requirements and views of the final product. One special case of software products that enterprise customers often need to support their day-to-day business are internal portals. By its formal definition, an enterprise web portal is a web-based interface for enterprise users which provides access to a broad array of information such as corporate databases, messages, document storage, internal applications and similar [2]. Portals are web applications that represent the company's central point for information sharing in its internal network.

When developing such applications, software engineers usually choose two approaches: development of a custom portal from scratch or combination of existing solutions such as content management systems (CMS) tools and document management systems (DMS). Each choice has its own advantages and disadvantages. In the case of custom portals, the developers are in total control of every fragment and functionality of the system. There is no need to spend time to learn documentation, source code and tutorials written by someone else. However, this approach can be rather time consuming as every little feature needs to be implemented and tested. By embracing out of the box solutions such as content management systems, programming time is drastically shortened, and instead the whole system need to be configured to suit our needs via various configuration forms and files which is preferred in some cases. The down side of this approach is that often knowledge of the inner workings of the system is needed. Another con is that customization options are not as generous as with our own developed solution, so the system of choice needs to be able to support the project requirements.

In this paper we present an integration of the Liferay [3] portal development platform and the Alfresco [4, 5] document management system [6] as flexible and easy to learn solutions for enterprise portal development.

Although both Liferay and Alfresco offer user activities tracking services that display basic user actions, those logs are only available as separate interfaces. Since our efforts are aimed at providing seamless integration of both platforms, an essential requirement is that user activities could be uniformly shown in one list from both Liferay and Alfresco. To achieve this goal, we developed a custom Liferay extension that gathers user activity logs from both platforms and displays them in chronological order on the desired portal page.

The paper begins with a brief overview of both platforms with Liferay covered in section 2 alongside its extension options discussed in subsection 2.1. Alfresco and its extension opportunities are covered in sections 3 and 3.1, respectively. The rest of the paper is structured as follows: in section 4 we present a solution for basic Liferay and Alfresco integration. In section 5 we discuss a way of browsing the Alfresco document repository trough out of the box Liferay portlets and we show some customization options. Section 6 discusses a problem of combining user activity logs from both platforms and showing it in the custom Liferay portlet, while section 7 provides the conclusion.

2. LIFERAY PORTAL

Liferay is the open-source web development tool developed in the Java programming language. Described as a content management system, Liferay is distributed in two different editions:

1. **Liferay Portal Community Edition** - A version with the latest features and supported by an active community. Distributed under the GNU Lesser General Public License.
2. **Liferay Portal Enterprise Edition** - A commercial version that includes additional services, updates and comes with full support.

Both editions can be downloaded bundled with a web server application. Liferay core functionality is its built-in content management system specialized for intranet and extranet portal web development, but latest installments of the platforms offer additional features beyond the basic content management. These features include integrated document repository with advanced document management functionalities, out of the box message boards, wiki system, etc. Basic Liferay web applications can be assembled using standard HTML pages that consist of various portlets and basic web content without the need for prior programming knowledge. Portlets [8] represent web applications that occupy portions of a web page and provide their own set of functionalities and data. Each portal page consists mainly of such portlets. Liferay portal comes with a number of pre-installed core portlets and themes while additional ones can be downloaded via the Liferay marketplace [9].

2.1. LIFRERAY PORTAL EXTENSION OPTIONS

For the advanced extension of the core functionalities, Liferay portal provides its own development tool called Liferay IDE. Liferay IDE is customized installation of Eclipse Java IDE with pre-installed set of Liferay plugins which enable development of the Liferay platform extensions [7]. Liferay IDE supports development of 5 types of Liferay extension projects:

1. **Portlet** - Provides basic file structure and libraries for development of Liferay portlets from scratch.
2. **Hook** - Hook is the Liferay extension used to catch portal life-cycle events and to override default actions performed for those events.
3. **Ext** - Ext plug-in present the programmatic extensions of the Liferay core functionalities.
4. **Layout** - By creating a layout project in IDE, developers are provided with the graphical editor using which custom page layouts can be defined. Layouts specify positions on which portlets can be placed on pages.
5. **Theme** - Enables development of custom portal themes including custom images, color schemes, CSS rules and JavaScript code.

Upon creation of a Liferay project, the developer is provided with a basic structure of the chosen extension project which includes standard configuration and resource files organized in a corresponding directory structure along with core libraries.

Even though the Liferay portal and its extensions are mainly written in the Java programming language, some feature implementations span multiple programming languages and frameworks. Core functionalities are implemented in the Java language, while data is displayed using the Freemarker and JSP template engines. Client-side functionalities are developed in the JavaScript programming language with support for PHP and Ruby portlets. Besides that, developers are free to use arbitrary libraries and tools in their projects. Finished projects can be automatically exported in an archive format using built-in IDE scripts. Deployment of portal extension includes exporting project as a WAR archive and uploading it to the Liferay's deploy directory. Upon upload, extensions get hot deployed and are ready to be used within seconds.

Besides developing custom extensions, the Liferay portal can be customized by configuring the portal and core portlets properties via the portal control panel or configuration files.

3. ALFRESCO

Alfresco is a free/libre Enterprise management system (EMS) [10] written in Java, available in 3 editions:

1. **Alfresco Community Edition** - free and open edition, with scalability and availability limitations, distributed under LGPL license.
2. **Alfresco Enterprise Edition** - commercially and proprietary licensed edition. Its design is geared towards users who require a high degree of modularity and scalable performance.
3. **Alfresco Cloud Edition** - SaaS version of Alfresco.

Alfresco can be downloaded bundled with Apache Tomcat or JBoss application servers.

Developed mainly as a document management system, Alfresco has grown into a fully grown EMS providing features such as [10]:

- Document management - including a built-in document repository and providing advanced management options.
- Web content management
- Repository-level versioning
- Repository access via CIFS/SMB/WebDav/ NFS and CMIS
- Lucene search
- Desktop integration with main office suits
- Clustering support

3.1. ALFRESCO EXTENSION OPTIONS

As access to all repositories is done using the Share application, the recommended way of extending Alfresco features is by extending Share. A possible way of customizing the core Alfresco functionalities is by modifying its source code. Some examples of this approach are shown in [10, 11, 12, 13]. Extending the Alfresco functionalities this way is done by using the Alfresco SDK which contains tools and libraries needed to start developing custom Alfresco plugins and extensions using Eclipse or NetBeans Java IDE.

In recent versions of Alfresco, the preferred way of developing custom extensions is using the Alfresco

WEB Script API [14]. By using the WEB Script API, developers can access repository features by invoking web scripts. Web scripts are light-weight web services used to perform custom actions on the Alfresco document repository. Alfresco comes with a predefined set of web scripts. Each web script is uniquely accessed by its URL which contains the script name and any optional parameters passed from a user.

Developing a custom webscript is relatively easy. Each script needs to have at least 3 required files:

1. **Script description file** - an XML file which provides basic information. The minimal set of XML elements in the description file includes:

   <*shortname*> - Script name

   <*description*> - Script description

   <*url*> - URL address by which the script is accessed with parameters specified as `name={value}`

   <*authentication*> - User authentication level required for script actions, available values are: *user*, *guest* and *admin*.
2. **Script implementation file** - a JavaScript file that contains the script action implementation. Besides the standard JavaScript features, various Alfresco objects and methods are available through the Webscript API. The request parameter values can be accessed by their name provided in the description file.
3. **Script template file(s)** - each script can expose action results in the HTML, RSS or JSON format, therefore each of the resulting formats needs to have its own template file with the corresponding extension. Objects are passed from the implementation files as attributes of the *Model* class instance and accessed by the name in the template.

Deploying the web script is done by copying these files into one of the Alfresco repository directories reserved for web scripts and refreshing the script index page after which deployed script appears in the available scrip list

4. BASIC INTEGRATION

By the basic Liferay-Alfresco integration we consider the implementation of two mechanisms:

1. Configuration of both Liferay and Alfresco instances to run on one application server instance.
2. Enabling the single sign-on (SSO) [15] mechanisms between them.

Running both platforms on one server instance reduces server machine load, improves mobility and maintenance of the whole system. By this form of integration we are basically creating one platform with a clean install that is easily reusable. The SSO mechanism increases portal safety and user experience by incorporating single authentication point for all portal applications. With SSO, users don't have to enter their credentials on every application access which can be highly error prone or store or remember multiple username/password combinations for each restricted part of the portal which effects system security.

Once deployed as the Liferay web application, Alfresco makes a selection of its portlets available in the Liferay portlets section. These portlets are:

1. **Repository browser** - enables browsing through the whole Alfresco document repository.
2. **Site document library** - used for browsing the document repository associated to the specific Alfresco site. The Alfresco document repository is divided into sites each with its own document and user spaces.
3. **Advanced search** - Provides the search functionality with wide range of search filters.
4. **My document libraries** - Provides a list of all document libraries (associated with sites) available to browse.

The user interface and functionalities in great deal resemble ones from the Alfresco Share application, so users have a large number of actions available through them. Only the advanced management and administration functionalities are omitted.

When a page with one of the browsing Share portlets is loaded, the portlet points to the root of the selected document space. If we want to point our pages and their corresponding portlets to link to a specific directory, we need to add an additional path segment to the page URL in Liferay. Format of this segment is:

```
#filter=path/path/to/directory
```

Where `path/to/directory` is path to directory in the Alfresco repository starting from the root of its library.

5. COMBINING USER ACTIVITY LOG

Providing a transparent insight into common user activities is an important feature of enterprise portals. By enabling users to see which pages have been created or modified, or which documents have been uploaded by their coworkers increases information flow and provides information of every company organization unit’s daily activities to everyone. Liferay and Alfresco both have its own generic user tracking services, but obtaining an automatically generated and integrated log has proven to be a challenging task.

Liferay tracks every user action using its User Activity service. It logs user activities from all portal

applications and provides a detailed report via the corresponding portlet. The Activities portlet provides a list of tracked user actions which contains information about users who performed an action, with the activity description and timestamp. Although this mechanism is intuitive and user-friendly, it can only be configured to show a selected number of activities, and there is no way to pick which type of activity will be shown to users. Besides that, a page creation and modification activities are not logged, even though they represent one of the most important events in the portal. A slightly more flexible way of displaying tracked activities is to use the Liferay's Asset publisher portlet. It can display activities related to each portal asset and filter them by a multiple criteria, including asset type. However, the event Asset publisher doesn't record page related activities.

Alfresco also offers two ways of logging user actions on the document repository. One is the activities dashlet available on the user's home page in the Share application. It is very similar to the Liferay Activities portlet and displays a list of recent Activities that can be filtered by users, a time period and sites within the repository. A more advanced way of trailing repository activities is the Alfresco auditing mechanism [16]. It represents the Alfresco service used to record and query user actions on the repository content which is very flexible and feature-rich. Since auditing provides a wide range of customization options and features, it is also more difficult to use and it represents a full-grown framework with its own API and libraries.

Since our goal is to integrate both platforms in one web portal it would be very useful to have an integrated user activity log which would display basic user actions from both the Liferay portal and the Alfresco document repository. Unfortunately, to this day, no solution has emerged from either Alfresco or the Liferay community. In the next subsection we will discuss the development of a custom Liferay portlet which collects and displays such an activity list.

5.1 CUSTOM PORTLET FOR INTEGRATED ACTIVIY LOG

The custom portlet needs to provide a combined list of user activities from both the Liferay portal and the Alfresco document repository and represent them in chronological order and in a uniform format. First problem that needs to be addressed is Liferay's lack of trails for page activities. Since page creation and modifications need to be transparent to all users, we need to provide a way to automatically capture and record these activities. Probably the easiest and most preferred way of achieving this is by implementing the custom Liferay hook extension. As described before, hooks can capture certain portal lifecycle events, so by implementing a hook on a desired event we can write an arbitrary programmatic reaction to it. For the purposes of capturing the page creation and modification events, we created a hook that implements the Liferay `ModelListener` interface which provides the methods that react to events in the `Layout` class lifecycle. The `Layout` class is Liferay's representation of the portal page. Outline of the `LayoutHook` class is shown in Listing 1.

As can be noticed, the interface captures all the important events in the portal page lifecycle, but since we are only interested in recording the page's creation, modification and destruction events, only the `onAfterCreate`, `onAfterRemove` and `onAfterUpdate` methods need to be implemented with the code that persists information about the performed action in the database. The hook is then attached to the desired event by creating a custom portal configuration file with the following entry:

```
value.object.listener.com.liferay.portal.model.Layout=
hooks.LayoutHook
```

Since Liferay relies on its service mechanism for storing data in the database, if we want to create tables for our custom entities and persist them, we need to build a service layer which will provide the basic CRUD operations. Building a custom service layer is easily doable using the Liferay IDE's graphic service builder tool. All we need to do is create an empty configuration file called *service.xml* and the graphic

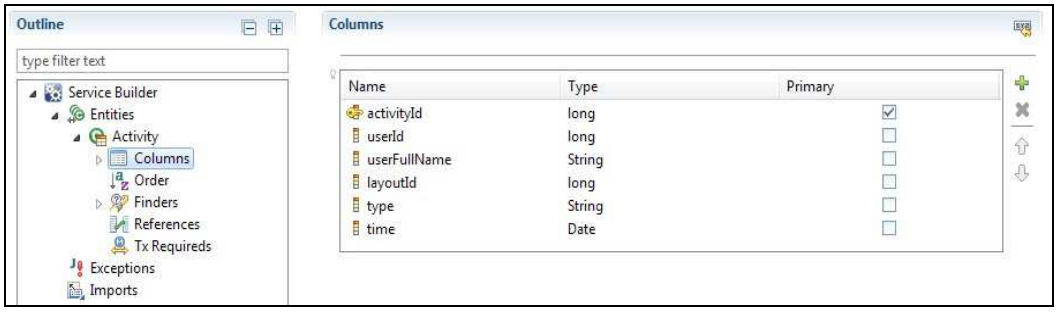

Figure 1. Custom entity shown in Liferay Service builder

interface that generates its contents becomes available. For our solution, we created an entity called Activity which is used to store the page related user activity information in the database.

The service builder graphic editor with our custom entity is shown in Figure 1. It provides graphic controls for defining custom entities that will be stored in the corresponding database tables and their associations. The *service.xml* file is simultaneously populated with the corresponding entries for each entity.

```
public class DDORLayoutHook implements
ModelListener<Layout> {

  @Override
  public void onAfterAddAssociation() {}

  @Override
  public void onAfterCreate(Layout arg0) {}

  @Override
  public void onBeforeRemove(Layout arg0) {}

  @Override
  public void onAfterUpdate(Layout arg0) {}

  @Override
  public void onAfterRemove() {}

  @Override
  public void onAfterRemoveAssociation() {}

  @Override
  public void onBeforeAddAssociation() {}

  @Override
  public void onBeforeCreate(Layout arg0) {}

  @Override
  public void onBeforeRemoveAssociation() {}

   @Override
  public void onBeforeUpdate()  {}
}
```

Listing 1. `LayoutHook` class outline

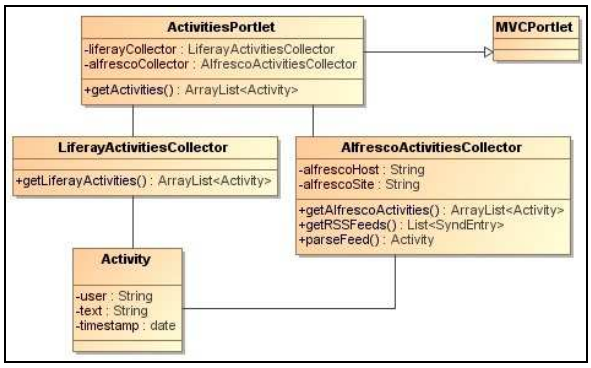

Figure 2. Activities portlet class diagram

With the service layer built to store activity data, and the hook implemented to catch page actions, we are able to capture user activities associated with portal pages, a feature that is missing in the Liferay core activity portlet. We then build our own portlet, which would read stored data along with the Alfresco document activities. Since only events that are of importance in this case are document and folder creation, modification and destruction dates, Alfresco audit proves to be too big and complicated, and the best thing to do is to find the way to access the Share Activities dashlet and collect its data. This dashlet provides an option for the RSS subscriptions, so all we need to do is access its feed from our portlet and parse its entries in the desired way.

Model of our portlet is shown on Figure 2.

Main portlet classes are:

- **`Activity`** - represents the portal user activity, it contains the activity description, full name of the user that performed the action and date and time when the activity happened.
- **`LiferayActivityCollector`** - class is used to collect data stored by `LayoutHook`. It simply queries the activity table and converts each row to the corresponding activity instance. It is important to note that to be able to access our custom service layer classes, our portlet needs to reference service JAR file generated by the service builder.
- **`AlfrescoActivityCollector`** - this collector class reads the Alfresco user activity RSS feed. Information about the Alfresco server and site within the repository needed to construct the RSS URL are stored and read from the portal configuration files. With parameters obtained from configuration files, webscript URL is constructed as follows:

  `http://hostname/share/feedservice/components/dashlets/activities/list?format=atomfeed&mode=site&site=siteName&dateFilter=28&userFilter=all`

  Where *hostname* is the name or IP address of the Alfresco server followed by the port number and *sitename* is a name of the site within the Alfresco repository from which we want to get activities. After obtaining the feed list, each entry is parsed and the `Activity` class instances are constructed from the extracted data. Parsing the RSS feeds can be done using any third-party library.

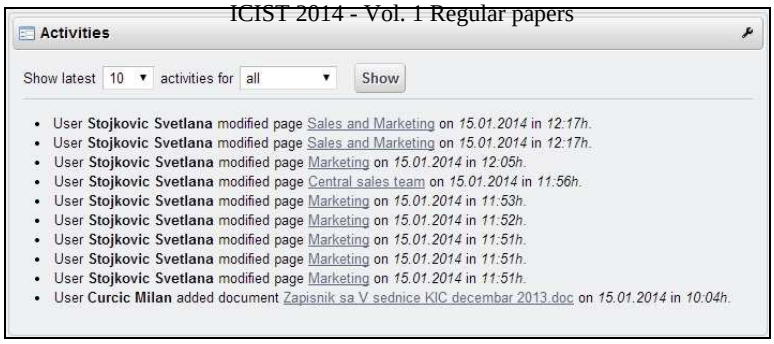

Figure 3. Custom activities portlet

- **`ActivityPortlet`** – the main portlet class, that extends the Liferay's core `MVCPortlet` class. Besides implementing the standard portlet functionalities, this class instantiates collector classes and calls their corresponding methods that return a lists with collected activities. It then joins the obtained lists and sorts activities by date so the most recent activity is shown first.

In addition to parsing Alfresco feed entries to extract actions, users and document names, we also need to convert links to documents and directories from feeds, so they are shown in Alfresco portlets on the portal instead of the Share application.

The activity portlet class constructs and sorts the unified activity list and passes it to the corresponding JSP template that is used to display data in the portlet section on the portal page. Since both collector classes convert collected data to the `Activity` instances, data from both sources are treated uniformly. The activity portlet can further be extended by interface controls for filtering and limiting displayed activities based on any user-defined criteria. The deployed portlet with example activities is shown on Figure 3.

6. CONCLUSION

By integrating Liferay's flagship portal and the industry-standard document management system we are getting best of both worlds: a proven Java-based portal framework paired with the best open source ECM. In this paper we discussed some options in integrating the Liferay portal and Alfresco into one powerful enterprise portal platform by extracting the best features from both systems and enriching it by custom made extensions to bridge the gaps. We presented an example of a tailored-made Liferay portlet and provided information needed to develop Alfresco web scripts which represent the backbone of our approach. This forms a communication channel in which Liferay portlets can call Alfresco scripts and display combined results seamlessly. We hope the paper presented enough information for developers who decide to follow this approach to be able to derive their own solutions that will suit specific requirements.

7. REFERENCES

[1] G. Milosavljević, B. Perišić, "A Method and a tool for rapid prototyping of large-scale business information systems", Computer Science And Information Systems, Vol. 02, pp. 57-82, 2004.

[2] M P Ahmed Hasan - The liferay Cookbook

[3] Liferay Portal, www.liferay.com

[4] Alfresco, www.alfresco.com

[5] Shariff, M., Alfresco – Enterprise Content Management Implementation, Packt Publishing, 2006.

[6] Gostojić, S., Sladić, G., Vidaković M., "Arhiviranje dokumenata u Alfresco sistemu", Zbornik radova YUInfo 2009, Kopaonik, Srbija, 2009. (in Serbian)

[7] Liferay Portal 6.1 - Developers Guide, https://www.liferay.com/documentation/liferay-portal/6.1/development

[8] Java Portlet Specification (JSR-168) , http://docs.liferay.com/portal/4.2/official/liferay-portlet-development-guide-4.2/multipage/ch01s02.html

[9] Liferay marketplace, http://www.liferay.com/marketplace

[10] Sladić, G., Gostojić, S., Milosavljević, B., and Konjović, Z., "Handling Structured Data in the Alfresco System", Proceedings of the International Conference on Information Society Technology and Management (ICIST), pp. 78-82, 2011. ISBN: 9788685525070

[11] Gostojić, S., Sladić, G., and Milosavljević, B., "Importing Document Hierarchy in the Alfresco System", Proceedings of the International Conference on Information Society Technology and Management (ICIST), pp. 88-91, 2011. ISBN: 9788685525070

[12] Pavić, I., Sladić, G., Milosavljević, B.,G, "Integracija upravljanja poslovnim procesima u Alfresco sistemu", Zbornik radova YUInfo 2010, Kopaonik, Srbija, 2010. (in Serbian

[13] Savić, G., Sladić, G., Milosavljević, B., "On-line uređivanje dokumenata u sistemu Alfresco", Zbornik radova YUInfo 2008, Kopaonik, Srbija, 2008 (in Serbian)

[14] Alfresco Web script API, http://wiki.alfresco.com/wiki/Web_Scripts

[15] Sladić, G., Zarić, M., Konjović, Z., Milosavljević, B., "Single Sign-On model za web aplikacije", Zbornik radova YUInfo 2008, Kopaonik, Srbija, 2008.

[16] Auditing, http://wiki.alfresco.com/wiki/Auditing_%28from_V3.4%29