Budapest University of Technology and Economics
Department of Measurement and Information Systems

The analysis and integration of open-source office productivity software in an enterprise environment

Author

Vajna Miklós

Advisors

Dr. Szentiványi Gábor, ULX
Dr. Szántó Iván, ULX
Horváth Ákos, MIT

2011.

Abstract

With the exponential growth of computer usage in the past two decades, almost all businesses are using electronic document processes. This requires integrated management of documents in processes, directly from office productivity software. These documents are part of a business workflow or a document management system, allowing easy collaboration.

The infrastructure of these processes is typically provided by a single vendor, where both the office software and the document management systems are from the same supplier. One of the biggest drawback of such a solution is that in almost all cases they are closed and proprietary – making it impossible to replace some of the elements with other alternatives.

Previous solutions include proprietary clients connecting to proprietary servers and open-source clients connecting to open-source servers, not allowing the usage of heterogeneous document management systems.

In this thesis, we integrate open-source office software to document processes in a way that they will be able to interoperate with closed, proprietary document management systems.

Kivonat

Az elektronikus dokumentumfolyamatok egyre szélesebb körben való elterjedése megköveteli, hogy a folyamaton keresztülhaladó dokumentumokat egységesen, integrált módon lehessen kezelni és módosítani közvetlenül, az irodai programokból. Ezek a dokumentumok üzleti folyamatok vagy dokumentummenedzsment rendszerek részei, lehetővé téve a könnyű kollaborációt.

Az ehhez szükséges infrastruktúra jellemzően egy beszállítótól származik, ahol mind az irodai programok, mind pedig a dokumentummenedzsment rendszerek is egyazon szereplőtől származnak. E megoldások egyik legnagyobb hátránya, hogy szinte kivétel nélkül zárt, tulajdonosi megközelítésre épülnek – kizárva annak lehetőségét, hogy egyes elemeit más, akár nyílt forráskódú programokra cseréljék.

Az eddig elérhető megoldások tisztán tulajdonosi vagy tisztán nyílt forráskódú megközelítést tettek lehetővé, nem engedve meg a heterogén dokumentummenedzsment rendszerek használatát.

Ebben a diplomatervben nyílt forráskódú irodai programokat integrálunk elektronikus dokumentumfolyamatokhoz oly módon, hogy azok képesek legyenek zárt, tulajdonosi dokumentummenedzsment rendszerekkel is együttműködni.

MŰEGYETEM 1782

DIPLOMATERV-FELADAT

Vajna Miklós (AYU9RZ)
szigorló mérnök informatikus hallgató részére

Nyílt forráskódú irodai programkomponensek vállalati környezetbe való integrációjának vizsgálata és implementációja

Az elektronikus dokumentumfolyamatok egyre szélesebb körben való elterjedése, megköveteli, hogy a folyamaton keresztülhaladó dokumentumokat egységesen, integrált módon lehessen kezelni és módosítani közvetlenül, az irodai programokból. E problémákra jellemezően csak az egy beszállítótól származó megközelítések nyújtanak megoldást, ahol mind az irodai programok, mind pedig a dokumentummenedzsment rendszerek is egyazon szereplőtől származtak. E megoldások egyik legnagyobb hátránya, hogy szinte kivétel nélkül zárt, tulajdonosi megközelítésekre épülnek kizárva annak lehetőségét, hogy egyes elemeit más, akár nyílt forráskódú programokra cseréljék.

A diploma célja, a nyílt forráskódú irodai programok integrálása elektronikus dokumentumfolyamatokhoz oly módon, hogy azok képesek legyen zárt, tulajdonosi dokumentummenedzsment rendszerekkel is együttműködni.

A hallgató feladatának a következőkre kell kiterjednie:

- Ismertesse a korszerű dokumentummenedzsment szerverek és munkafolyamat kezelő rendszerek felépítését.
- Tervezzen és valósítson meg egy megközelítést nyílt forráskódú irodai programkomponensek különböző dokumentummenedzsment szerverekhez való hatékony integrációjára.
- Készítsen egy megvalósítást nyílt forráskódú irodai programkomponensek munkafolyamatba történő integrációjára.
- Vizsgálja meg a megvalósított megközelítések alkalmazhatóságát több különböző platformon.

Tanszéki konzulens: Horváth Ákos
Külső konzulens: Dr. Szántó Iván és Dr. Szentiványi Gábor (ULX Kft.)

Budapest, 2011. március 12.

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Méréstechnika és Információs Rendszerek Tanszék

[illegible]

Dr. Horváth Gábor
tanszékvezető

Contents

Chapter 1

Introduction

Integration of an office productivity suite into electronic document flows can be divided into two parts: document management integration can be done alone, then on top of that, adding workflow support to an office client is possible.

1.1 Document management

Every major software development project uses some kind of version control system to track the source code of their projects. The introduction of such systems help multiple developers to synchronize their work, helps debugging (with retrieving older versions, if necessary) and provides a way to document changes.

There are different approaches to version control:

- Centralized (Subversion [1] is probably the most popular centralized solution) systems have a central server. A centralized system has a simple architecture, but on the other hand, it provides only a few features.
- Distributed systems operate in a distributed environment (Git [2] is a good example). Distributed systems can operate in more extreme conditions, while they are harder to learn, because of their steep learning curve.
- A radically different solution is a hybrid cloud-based approach: in that case there is a logical central server, where clients are in fact served by a large cluster, providing enhanced availability.

The need for tracking versions and collaborating on documents is not specific to software developers. An other type of similar software is usually referred to as a document management system. It focuses on project management and it is meant to be used by a wider range of users without specific software engineering knowledge. That means they usually do not manage software source code files, but documents created by office suites such as Microsoft Office [3] or LibreOffice [4] (formerly OpenOffice.org [5]).

There are numerous, currently unsolved problems in this area. Such enterprise systems are supposed to be modular: each module communicates with the other components using an open (documented) protocol and companies are allowed to replace one module with another implementation. In fact, this is one of the major reasons for using open-source software components in an enterprise environment: that way protocols are always open, and **companies can avoid vendor lock-in**.

Enterprise companies using proprietary software, but wanting to adopt open-source solutions can start the migration with server products. The benefit of starting with servers is that only the central infrastructure has to be modified. Given that the way of communication is unchanged, the clients can be left untouched. Another approach is to focus on the client-side, as important data is rarely stored there, so starting the migration with the client components avoids the risk of a potential data loss, in case software quality is lower than expected.

One of the most widely used document management servers is Microsoft SharePoint [6]. It is well integrated with Microsoft Office, but previously it was not possible to replace the office suite with an open-source alternative, because it did not support SharePoint. An open-source alternative of SharePoint is Alfresco [7], which provides similar features, but uses its own protocol.

Finally, in case of the cloud approach, the client is a simple web browser and even the editing software is downloaded from the server each time the user wants to edit the documents (Google Docs [8] is a good example). While this method is easy to use, it has its drawbacks as well. At the moment none of the cloud-based solutions have all features a rich client such as Microsoft Office or LibreOffice provides. Additionally, storing documents in the cloud requires a stable and fast network connection during the editing process (which is currently isn't guaranteed in many cases) and requires trust in the cloud provider when editing sensitive documents of a company.

In the current thesis, we are presenting a centralized client-side solution for using an existing, proprietary document management server with an open-source office productivity suite. My solution is an extension to LibreOffice that adds support for accessing and managing documents from a SharePoint server directly from the office suite.

1.2 Workflows

Document management systems are often used as part of document-based workflows. Workflows represent a collection of connected steps to achieve a certain goal. A workflow engine can manage process definitions storing these sequences. A process definition describes each step, and the connection between steps. The steps are represented by nodes inside a process definition. Once the definition is ready, it can instantiate a process definition, creating a process instance. Such an object then always has a precise state:

- One or more of its nodes are active, and
- It can have multiple associated variables.

This model then can help extract some of the business logic of applications into more abstract process definitions, described in a more separated, declarative way[1].

Using workflows to represent processes describing a document's change list is a common pattern. In short, they are referred to as document-based workflows. Most solutions are implemented with a strong dependency between the document management server and the workflow engine, which introduces the problem of vendor lock-in once again.

On the other hand, in case a decoupled document server / workflow engine architecture is found to be necessary – to address the problem of vendor lock-in –, the integration logic has to be implemented in the client. Given that there is already a planned document management client in the office suite running on the client, it's a natural extension to introduce workflow integration in the office client to manage document-based workflows.

Based on the experience from a real world use-case, we identified the following requirements as mandatory:

- If a document is open, it's useful if operations on that document can be done from the office suite, which already has a reference to the document. Think about starting a chosen process definition to start a new instance, associated with the current document.
- The workflow server talks about tasks and group of tasks, and each task is connected to a certain document. This is a conceptional difference to the model of folders and files, which is already familiar to a user.
- A common pattern is to assign a task to a group of users, and then require exactly one member of the group to perform the task.
- As outlined above, the ability to change the document server while not touching the workflow engine – and also the other way around – may be a requirement when planning a document-based workflow system.

[1]Even if script tasks can contain imperative code, the main structure of a process definition is always declarative.

- It can be necessary to limit the write access to the associated document when working on it to a certain part of the file. This helps the user performing the task focusing on the work item itself, even if the document is a large one.
- It is uncomfortable if storing the document on the server and making a decision as part of the current workflow task is not a single atomic operation.
- It is important to realise that while finishing the work on a document and marking a given workflow task is usually done together, sometimes only the previous is wished. Support for such incremental snapshots is vital.
- Finally, even if rolling back the workflow state is not trivial (given that the workflow interacts with external services, rolling back the workflow itself is not enough, external components should be notified using compensation actions), support for releasing claimed group tasks is required.

There is one more operation where a modified workflow server is needed. In general, but especially when group tasks or decisions are involved, creating audit logs of process and node instances are important. Being able to them from the office client makes viewing them much easier.

Based on that, we can build a list of functionality that can serve as a motivation – to help interacting with document-based workflows, in no particular order:

- Workspaces, documents, task types and tasks should be represented to the user following the well-known file-metaphor.
- The document server should be decoupled from the workflow server, to allow a modular architecture, having independent layers.
- Support for both personal and indirectly assigned group tasks is required.
- Even if group task is claimed by a user, the system should support releasing the task back to the group, if the user realises he/she is not able to perform it in time.
- It should be possible to associate a new process instance to the current document.
- If the current task involves making decisions, the list of possible choices should be retrieved from the workflow engine.
- Accessing the audit log from the office client is required.
- Supporting incremental snapshots till the task is marked as completed is needed.

- As a helper feature, masking the document so that only the relevant parts are editable by the user is useful.

To summarize our goals:

- We want to design document management integration on the client-side.
- We want to add workflow integration support to the solution.
- We want to keep the created solution portable across different platforms.

The rest of this thesis is structured as follows. First, we introduce necessary background knowledge, which was present before the current thesis, but is needed to understand the rest of this work (Chapter 2). In Chapter 3, we propose our approach. Chapter 4 describes the design of the solution, as well as relation to the underlying techniques. Next, we detail the implementation we created (Chapter 5) and also evaluate it (Chapter 6). Finally we present work related to the current solution (Chapter 7) and give a summary, including future development directions (Chapter 8).

Chapter 2

Background

In this chapter, we will have a look at solutions we want to build on in our approach. The first section details background of document management systems, the second one introduces workflow engines.

2.1 Document management

Once we defined what a document management system is, this section details standards and implementations of these systems. At the end of the section, we describe UNO, the component model we want to base our approach on.

2.1.1 General architecture of document management systems

A document management system has a typical client-server model: a document management server stores the documents in a document repository, which can be accessed via various interfaces. The other part of the system is accessed via a client, which has built-in support for opening, saving and editing documents.

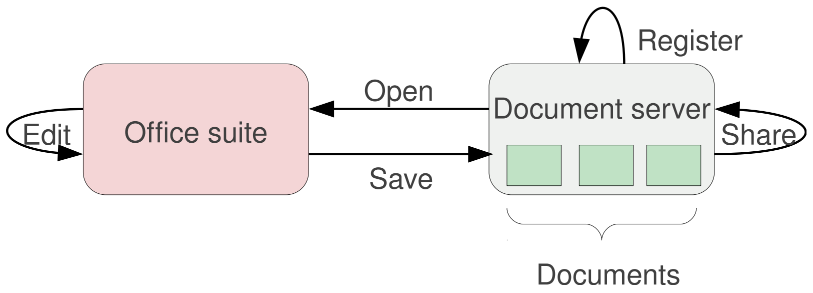


Figure 2.1. Architecture of a document management system

Figure 2.1 shows the two entities of the system. They have the following operations:

- The server primarily listens to client requests. Additionally, it allows performing operations directly on the server.
- The client, connecting to one or more document server.

Every user can have access to document **workspaces**. Workspaces can have documents, links and tasks:

- documents: file based document representation
- links: collection of bookmarks
- tasks: work items related to the documents of the workspace

A workspace can be shared with different permissions (read-only, read-write), and that is typically done by sending an invitation email which can be accepted by the other user.

A user may access the document server using a web browser, or via rich client applications. The advantage of the web browser interface is that it can be accessed from almost everywhere, however, document editing can't be performed. If such an operation should be performed, then the user has to manually download the document, edit then upload it. Done rarely, this does not cause a problem, but of course it is uncomfortable for daily work.

The other interface is a rich client, which is installed on the machine of the user. Vendors prefer to produce a corresponding client for their server, Microsoft SharePoint and Microsoft Office is a typical setup.

In case of servers or clients speaking different communication protocols, selection of the used protocol is selected differently on client and server side. Servers can listen on different addresses, and in this case the address identifies not only the server, but the used protocol as well.

For example, Alfresco, has its native protocol, but also (more or less) provides support for the SharePoint protocol. As a result, it can be configured to listen on one URL as an Alfresco server, and on an other URL as a SharePoint server.

Clients can have different extensions or plug-ins to handle different protocols. For example, Microsoft Office can accept SharePoint URL's in the standard open file dialog, while the Alfresco extension for OpenOffice.org has a dedicated menu in the application to connect to an Alfresco server.

It is also common for the client extensions to have minimal business logic. For example the proprietary SharePoint extension to OpenOffice.org, created by Oracle can't talk to every SharePoint server, but Microsoft Office does – as long as a server-side component provided by Oracle is not installed on the server. While this approach may be compelling during development, actually it is uncomfortable for system administrators.

2.1.2 Related standards

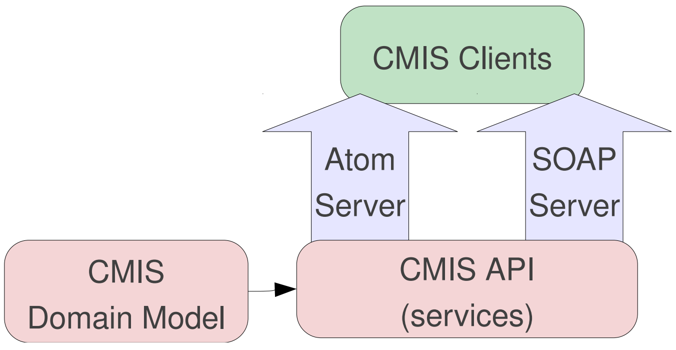


Figure 2.2. Architecture of the CMIS standard

The specification called Content Management Interoperability Services [9] is created to address compatibility issues between various document management server implementations. OASIS approved it on May 1, 2010. Also, at the time of writing, its implementation is far from complete among major document management servers. Alfresco 3.3+ implements the client side only, SharePoint 2007 does not support it yet, so at the moment it is only a vision that all major document servers will implement this specification.

On the other hand, there are a lot of implementations in other products, such as IBM Lotus Connections 3.0 [10] (server side only) or TYPO3 (client and server side).

CMIS has two main goals:

- Providing a list of web service (SOAP) interfaces, which are language-independent.
- Separating the service and content, making it possible to implement services for legacy document repositories without modifying them.

To realize this separation, it defines a domain model, describing data model elements, services to operate on this data model and concrete syntaxes: the Restful AtomPub Binding and the Web Services (SOAP) binding (see Figure 2.2).

2.1.3 Concrete implementations

In this subsection, we give a short comparison of Microsoft SharePoint 2007 and Alfresco Community Edition 3.4.

Alfresco claims to be “The Open Source Alternative for Enterprise Content Management”. [11] A difference is that Alfresco does not allow deletion wherever possible. Of course it is possible to delete documents, but it isn’t possible to delete document versions, to overwrite a previous document version or to delete a whole workspace as a single operation.

On the other hand, SharePoint uses its leading position to avoid publishing its communication protocol. This protocol was designed before the CMIS standard was published, and it is not a standard. For example the *put document* method has a *document* parameter – the documentation [12] does not mention the *timestamp of last modification*, part of the meta information structure, which is in fact mandatory (and there is no problem with that being mandatory, since it uses an optimistic concurrency approach). There is no similar problem with Alfresco, where in the worst case it is possible to check the source code or contact the Alfresco engineers using their community forums.

When evaluating properties, we selected the most common use-cases, required by enterprise companies. We intentionally did not evaluate management of document workspace permissions, links inside document workspaces and tasks inside document workspaces, as those features are beyond the scope of the current thesis.

The properties we evaluated are detailed in Table 2.1.

Feature	**SharePoint**	**Alfresco**
License	proprietary	open-source
Maturity	8 years [13]	6 years
Creating a document workspace	supported	supported
Deleting a document workspace	supported	not supported[1]
Checking out a document	supported	supported
Checking in a document	supported	supported
Cancelling the checkout of a document	supported	supported
Getting documents	supported	supported
Putting documents	supported	supported
Listing versions of a document	supported	supported
Viewing previous versions a document	supported	supported
Deleting previous versions a document	supported	not supported
Overwriting previous versions a document	supported	not supported
Restoring a previous versions a document	supported	supported
Deleting a document	supported	supported

[1] This feature is supported when using the native protocol of Alfresco.

Table 2.1. Comparison of SharePoint and Alfresco

To sum up, the key features we need to support are:

- management of workspaces, folders and documents using a folder/file metaphor

- version control of documents: the creation, update, list and delete operations on versions
- check-in / check-out support for documents

2.1.4 Differences from version control systems

Version control systems in general have richer semantics, more features to track software source code. In order to make document management systems easy to use for non-technical people, some of the features of version control systems are simply missing from document management systems. In this subsection, we describe the major removed operations.

Please note that this does not mean version control systems are superior. They are optimized to handle text based files. Version control systems are not efficient at handling binary files – for example zipped XML files, used by the ODF/OOXML formats – while document management systems are designed to deal with such formats.

The development of source code is typically not linear. It's common that branches are created and merged during development, while a document server forces you to follow a single development line.

Source code repositories are also checked out as a single operation, containing all files of the repository. Similarly, when a client checks in multiple files, that makes a single commit, and later it is possible to see all the files modified by that commit. Document servers make the assumption that the client wants to check out a single file, and a commit affects only a single file.

Documents are improving, but we rarely speak about document bugs, as we speak about software bugs. Because of that, version control systems usually provide a way to check out the state of the entire repository at a given earlier time, to discover which commit introduced a specific bug. Document servers allow listing of versions of a document, then the user has to manually select the version which is closest to a given date.

Finally, version control systems provide a way to annotate source code files: to determine the author of every single line of a file, where the author of a line is the person who authored the commit introducing the line in question. Document management servers do not pay attention to this, since the document formats they usually track (think of ODF or OOXML, again) provide a *track changes* feature already, so it makes little sense to duplicate that functionality on the server side. Not to mention that these document formats store the *model* of the document (model, as in Model-View-Controller), and when we speak about lines, we speak about the *view* of the model, so the same annotate operation would have a different meaning here.

To sum up, we can see that document management systems feature less operations in general to make usage by a wider user base possible – but we should not forget that version control systems solve a related, but still different problem, so neither of them is an ultimate tool, making the other one useless.

2.1.5 UNO compontents

As already introduced in section 1.1, my solution will be a LibreOffice extension that registers a UNO [14] component. UNO is the interface based component model of LibreOffice. The purpose of UNO is to abstract service definitions (defined in a language-idependent IDL-like syntax) from service implementations – they can be developed in multiple languages: C++, Python, Java or BASIC.

The caller is not aware which language the implementation uses, and it does not need to be, either. In our extension, we call BASIC macros from the menu elements, which invoke the underlying Java code, where the business logic is implemented.

All the Java code is contained by a single jar file, the workflow of creating it is shown in Figure 2.3.

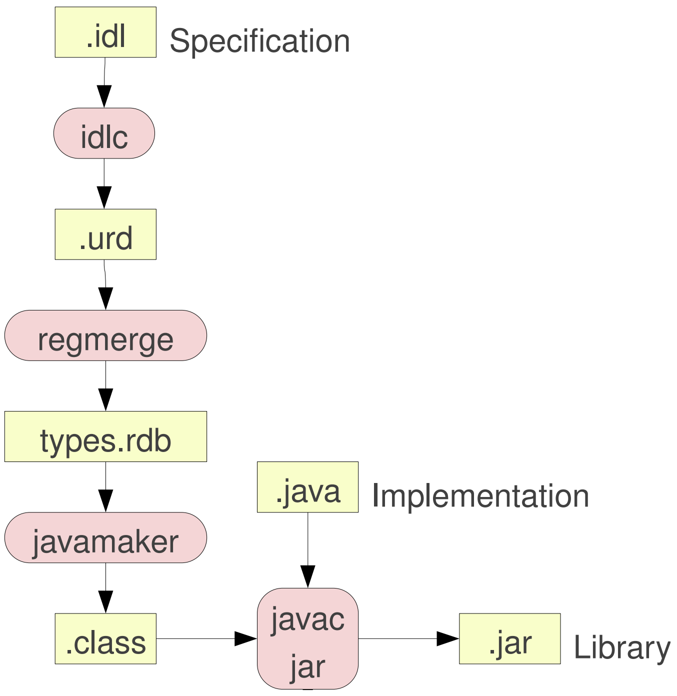


Figure 2.3. Workflow of implementing an UNO service in Java

- *idlc* compiles the .idl files to UNO Resource Descriptors (URD),
- *regmerge* merges all the .urd files of the component to a single Resource Database,
- *javamaker* creates interfaces (.class files) from the Resource Database,

- finally the standard Java development tools (*javac*, *jar*) create the final jar, including the Java implementation.

Once the jar package is ready, a zip archive is created, containing:

- *Addons.xcu*, containing the menu items
- the BASIC library (callbacks for the menu items)
- Java libraries used by the Java implementation
- the Java implementation
- metamodel of the stored settings
- other files: metadata, extension description, license, etc.

This zip archive gets a *.oxt* extension, and can be easily installed by the user, using the extension manager of LibreOffice.

2.2 Workflows

2.2.1 Application servers

The term application server is used to describe the software framework on which the business logic of an application runs. The application server typically interacts with SQL database servers, file servers or web servers.

Web application servers

A common usage of application servers is a web application server, where the framework handles common web-related tasks, so that application developers can focus on the development of the software project itself. Such common tasks include, but not limited to:

- handling connections towards the database layer
- handling incoming requests from the HTTP server
- management of a cluster
- load-balancing features
- fail-over capabilities

- centralized configuration
- security: cryptographic and authentication features

Given that the integration between a contained application and server framework is strong, it is a common practice to use the same programming language to implement both. As a consequence, e.g. Java applications usually run on application servers implemented in Java, and so on.

There are two approaches[1] to match this requirement. One approach is to use a generic web server, for example the Apache HTTP Server [15]. Given that almost all programming languages has a C API to interact with managed code, plugins for the web server can be implemented in C to execute managed code when an HTTP requests arrives, addressing the application in question.

The other approach is to have a dedicated HTTP server, which can only execute applications written in a given programming language, but it does so more effectively. Supported programming languages include:

- .NET: both Microsoft and 3rd-party application servers are available. Microsoft provides this functionality in their Windows Server product and their .NET Framework.
- Java: we will discuss this in detail in section 2.2.1.2.
- PHP: Zend Technologies produces Zend Server to host PHP applications.
- Ruby: the Ruby on Rails framework includes the WEBrick HTTP server.

Java application servers

Java application servers have the benefit of implementing common interfaces: that way changing application servers is supposed to be an easy operation.

These common interfaces live under the *javax.servlet* namespace [16]. Some of the included concepts:

- The *Servlet* interface: an implementing class runs within a web server. Its most used implementation is the *HttpServlet* class, that can handle HTTP GET, PUT, POST and DELETE requests.
- The application can be packaged to a *WAR* file before it is deployed on the application server.

[1] A third, but ineffective method is to execute the interpreter of the programming language each time a request is served, referred as Common Gateway Interface (CGI).

- Servlets are invoked by a *web container*, the module of the application server directly interacting with servlets.
- Template-based output can be generated using *JavaServer Pages* (JSP). Such templates are processed by the JavaServer Pages compiler.

Popular Java application server implementations include:

- GlassFish: An open-source Java application server, started by Sun Microsystems for the Java EE platform, and it is still supported by Oracle. The name of the product with commercial support is Oracle GlassFish Server.
- JBoss Application Server: see detailed discussion in section 2.2.1.3.
- Apache Tomcat: This is the application server implemented by the Apache Software foundation, providing an implementation of the Java Servlet and JavaServer Pages specifications.
- IBM WebSphere Application Server (WAS): A commercial Java application server from IBM.

JBoss

JBoss Application server (in short, JBoss AS) is an open-source application server based on Java EE, originally developed by JBoss, Inc – now a division of Red Hat. Besides implementing a server on top of Java, it implements the Java EE specification as well. JBoss is written in pure Java, meaning that it is usable on any operating system Java runs on.

At the time of writing, the latest stable version of JBoss is AS7, which introduces a rewritten codebase, focus on parallel execution, and radical speedup of start-up time [17].

JBoss provides the common application server features, including:

- clustering and load-balancing
- fail-over, even if sessions are configured
- an implementation of the Authentication and Authorization Service (JAAS) specification
- management and monitoring though Java Management Extensions (JMX)
- implementation of the Java Persistence API (JPA), though Hibernate
- implementation of JavaServer Pages and / Java Servlet, through Tomcat
- integrated support for Java Database Connectivity (JDBC)

- OSGi support, which means one can deploy OSGi bundles in a JBoss server, next to other existing non-OSGi deployments

JBoss itself is open-source, though commercial support is available from Red Hat.

2.2.2 Business Process Model and Notation

What is business process modelling?

Business process modelling is a process of three steps:

- find nodes: have a look at the real-world process and split it to a sequence of individual steps
- node properties: define additional properties for nodes, such as the description or due date
- transition between nodes: once transitions are declared, we can have an executable process definition

The Business Process Model and Notation (BPMN) standard [18] supports all these steps, Figure 2.4 shows such a completely defined process definition, using the Eclipse tool of jBPM.

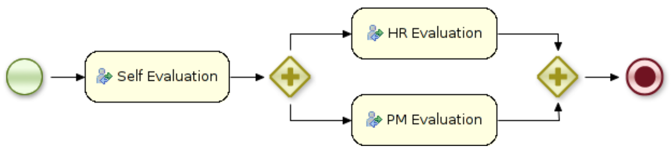


Figure 2.4. A simple BPMN process with three human tasks and two gateways

The rest of this subsection gives a brief introduction to BPMN concepts used in the later chapters of this thesis. Due to size limitations, we didn't consider describing all types of each concept here.

BPMN basic concepts

The most important elements in a process definition are nodes, and transitions between these nodes. Figure 2.5 shows that BPMN uses the following concepts to represent these:

- activities: a step of the real-word process – for example a human task or a script task
- events: represent the start and end of a process, crossing a deadline or error handling
- gateways: diverging and converging gateways model parallelism and choices

- connectors: declare valid transitions between elements

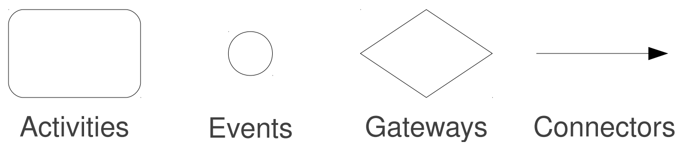


Figure 2.5. The basic diagram elements of BPMN

Figure 2.6 shows that there are two types of activities: tasks and subprocesses.

- tasks: are part of a process definition that are not described in more detail
- subprocesses: refer to reusable independent parts of process definitions

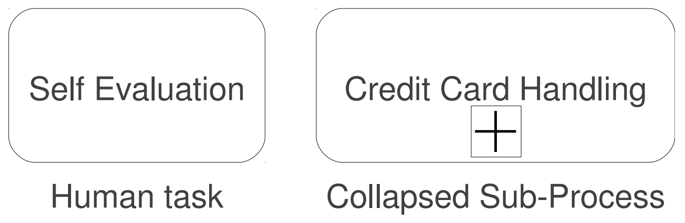


Figure 2.6. Activity types in BPMN

BPMN supports start, intermediate and stop events. Start events mark where the execution of the process will begin. Figure 2.7 shows three start even types:

- normal start: used when an explicit start invokes the process
- message start: indicates the process was triggered by receiving a message
- rule start: means the process was activated by a business rule

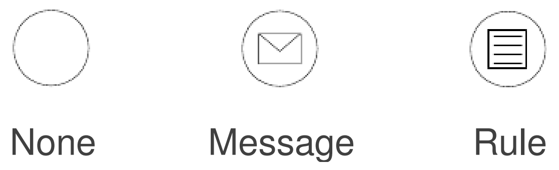


Figure 2.7. Three start event types in BPMN

Intermediate events can occur anytime after the start, but before the end. They can be (see Figure 2.8):

- normal events: are generated by activities

- timer events: can occur when crossing deadlines
- compensation events: can occur when rolling back a transaction

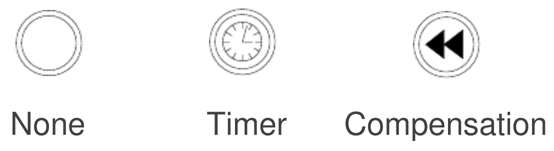


Figure 2.8. Three intermediate event types in BPMN

Finally, end events indicate that the main process ended. Subtypes shown on Figure 2.9:

- none end: means the main process ended, but subprocesses are still running
- terminate end: indicates the execution of the process instance ends
- error end: describes something unexpected happened

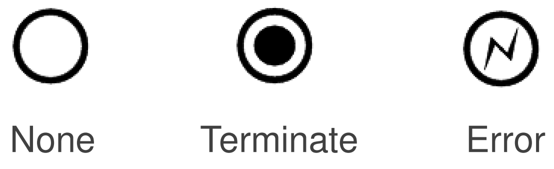


Figure 2.9. Three end event types in BPMN

The two most common used gateway types (Figure 2.10):

- exclusive gateway (can also be marked with an internal “X”): only one branch is executed
- parallel gateway: branches are executed in parallel

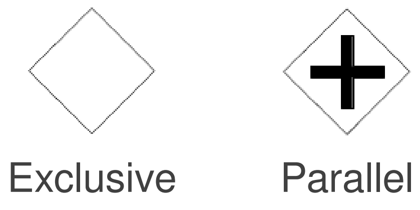


Figure 2.10. Two gateway types in BPMN

The last BPMN type we introduce here are connectors. Three types of them can be seen in Figure 2.11:

- sequence flow: is used for ordering between tasks
- message flow: indicates two activities are prepared to send / receive messages
- association: is used to assign additional data to activities

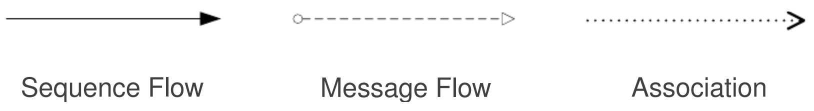


Figure 2.11. Three connector types in BPMN

BPMN support in jBPM

3.x series of jBPM and earlier supported the jPDL format for declaring process definitions. 4.x series of jBPM introduced support for BPMN, while jBPM 5.x supports BPMN exclusively. Tools are available to somewhat automate the conversion from jPDL to BPMN.

2.2.3 Workflow engines

What is a workflow engine

As already mentioned in the *Introduction* chapter, a workflow engine manages process definitions and instances. The primary functionality the engine itself provides is the ability to execute these instances, storing the state of the process instance in a permanent database. Once the process is finished, it becomes a *historic* process instance, which is still stored.

A process definition consists of nodes. There are two node classes:

- human tasks are to be executed manually
- script tasks, diverging or converging gateways are executed automatically

Additionally, each node can have several properties, for example imperative code attached to a script task or due date for a human task.

The architecture of a workflow engine is event-based, changes to process instances are triggered by events:

- internal events: when a deadline is reached
- external events: when a task is completed, a trigger in an other application starts a new process instance, etc.

When executing actions for external events, the following steps are implemented:

- Check if the requested action is valid in the active state (i.e. there is a transition from the current state to the requested state in the process definition).
- Authorize the caller to determine if executing the action is permitted by the user.
- Executing the action itself: once the operation finished, act based on its exit code (initiate error handling if necessary).

There are three popular standards to describe a process definition:

- The Business Process Execution Language (BPEL): is a standard describing an executable language where the business process interacts with web services.
- The Business Process Model and Notation (BPMN): will discuss it in subsection 2.2.2.
- The jBPM Process Definition Language (JPDL): a proprietary language developed by JBoss, focusing on readability.

Operation modes

A workflow engine can have two different operation modes. The first follows a client-server model. Figure 2.12 shows this situation, where the workflow engine is behind a server interface.

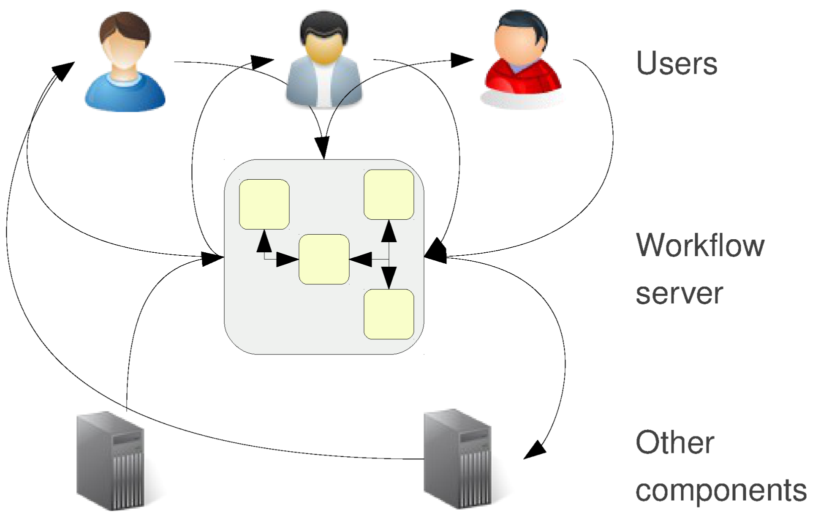


Figure 2.12. A workflow engine in server mode

For example, in case of jBPM there is a REST API available for clients to interact with the engine. The benefit is that the workflow server can run on a separate machine, also there is no need to start or stop it when clients start or stop. Additionally, multiple clients can connect to the workflow server at the same time.

Figure 2.13 describes the second, embedded mode.

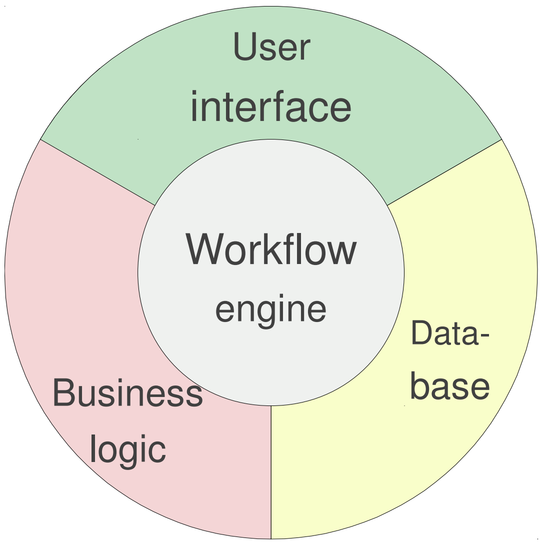


Figure 2.13. A workflow engine in embedded mode

In this case the application and the workflow server can run in the same process, the workflow engine is used as a simple library, which produces increased performance.

jBPM

Implemented in Java, jBPM [19] is an open-source business process manager. One property that makes it different from other workflow engines is that it tries to give better support for developers as well. Usually workflow engines target non-technical people, making the life of users already used to imperative programming hard.

The core supports executing processes defined according to the BPMN 2.0 specification[2]. On top of that, it offers more:

- a web-based and an Eclipse plugin to design process definitions using an easy drag and drop method
- its storage back-end can be any database which is supported by JPA[20]
- handling of human tasks is a separate component – a sample implementation is included in the release
- a BPM console (detailed later in subsection 2.2.5), built on top of the core provides web-based management and a monitoring interface for process instances and tasks
- a repository, named Guvnor, can be optionally used to deploy processes to

[2]Previously it only supported the jPDL language.

- the storage backend can log all detail needed for audit functionality

Script tasks can be implemented in multiple imperative languages:

- MVFLEX Expression Language (MVEL): a hybrid dynamically/statically typed scripting language
- Java: in case a non-scripting language is preferred.

2.2.4 Object-Relational Mapping

Both jBPM itself and its sample human task server implementation uses object-relational mapping (ORM) to store data in relational database, accessed using SQL. The general problem is that object-oriented applications store runtime data in objects, while a persistent data store can't store object as-is, code has to be manually written to store and restore object to / from a relational database.

An ORM automates this procedure, reducing code needed to be implemented manually. A disadvantage of using an ORM can be that highly optimised hand-constructed queries are faster then the ones auto-generated by the ORM. A solution for this can be the usage of stored procedures, in case portability between storage back-ends is not a requirement.

In case of Java, the de facto ORM tool is hibernate, jBPM uses that as well. Hibernate introduces the Hibernate Query Language (HQL), which is fully-object oriented: on one hand that means it understands inheritance and polymorphism, on the other hand it still provides flexibility similar to the traditional stored procedures.

2.2.5 The BPM Console

The BPM Console is a frontend to jBPM, allowing the usage of the workflow engine in a client-server model. Here we introduce concepts which will be later necessary to understand how we extended this project to meet our needs.

Technical overview

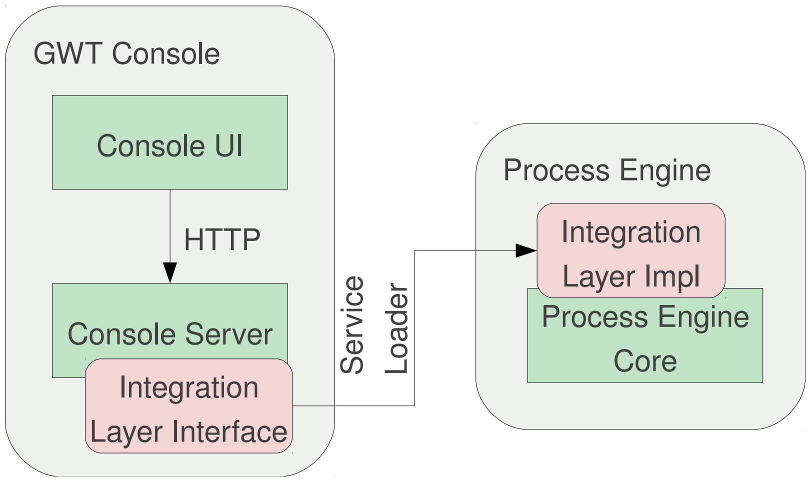


Figure 2.14. Components of the BPM console

The BPM Console is a generic web console for process engines. In practice it is used by:

- Riftsaw: a BPEL engine
- jBPM: a BPMN engine
- Drools: a business logic integration platform

Components are shown on Figure 2.14:

- the integration layer is an interface to be implemented by process engines
- the console server provides a REST API and translates HTTP requests to the integration layer
- the web console is a Google Web Toolkit (GWT) application communicating with the console server only

The integration layer discovers the available implementation using the standard J2EE service loader mechanism [21]. The console-related artifacts are described in Table 2.2.

Archive name	**Description**
gwt-console.war	The user interface
gwt-console-server.war	The REST server
gwt-console-rpc.jar	Domain model
gwt-console-server-integration.jar	Integration layer

Table 2.2. Artifacts of the BPM console

Workspace framework

The workspace of the BPM console is a concept completely independent from the document server workspace. The workspace here is the user interface of the web console. The workspace features editors. These editors are implementing a common API, and they serve two purposes:

- they can be used for management of the workflow engine
- they serve as an example on how to use the console server

The editor interface implementations are called *buildtime plugins*. For our purposes, two plugins are interesting:

- *ProcessEditor*: can be used to edit process instances: start, inspect or terminate
- *TaskEditor*: can be used to complete, claim, release tasks

Both of these editors are specific to jBPM, the list of loaded plugins is determined at build-time: a *profile* can be specified as a parameter of the build process, and the jBPM profile lists these plugins. As a result, the selected profile becomes the *workspace configuration*.

The console server supports *runtime plugins*. These are:

- FormDispatcherPlugin: handles the completion of task forms
- GraphViewerPlugin: handles rendering of the current state of a process instance
- ProcessEnginePlugin: handles management of the process definitions

The same service loader mechanism is used to load the implementation of these interfaces as for the integration layer, however, in case these are not available, then the related functionality is simply hidden from the web interface, it does not cause a fatal error.

Management capabilities

Management capabilities are determined by the installed editor plugins. The project suggests that they try to keep a balance between:

- providing plugins which are examples only
- supporting every corner case

The process editor offers the following features:

- Management of the life cycle of process instances: process instances can be started, terminated or deleted. Termination ends the process instance, deletion also removes the history information.
- Visualization of process activity: the already mentioned *GraphViewerPlugin* can draw the graphical representation of the BPMN definition and show the actual instance state.
- Instance data can be inspected: variables associated with a given process instance can be accessed read-only.
- Process form handling: if the process definition has a form associated (which is the way to start parametrized workflows), it will show it and let the user fill it out before the process instance starts.

The task editor can manage tasks of the currently logged in user. It features:

- a personal and a group task lister
- a manager for the task life cycle: it can be open or assigned (whenever it is completed, it gets removed from the list, so there is no such state in the user interface)
- a handler for task forms: whenever the task has an associated form, the user can provide input or review read-only data

In this chapter, we had a look at technologies we want to use in our solution. We saw how a document management system looks like, what UNO is, introduced BPMN, jBPM and its BPM console. Chapter 3 will give an overview of our solution, using these concepts.

Chapter 3

Overview of the Approach

Our approach is to take an existing open-source office suite, and extend it to be able to communicate with the document management server. Once this is ready, we can connect to an existing workflow server as well, extending it, if necessary.

3.1 Document management

In the current thesis, we propose a client-based approach for the integration of open-source productivity software into enterprise document management systems. We discuss the case here when an enterprise company tries to replace the client part of the document management system with an open-source alternative, while keeping the existing proprietary server-side part of the system. As a consequence, the center of our approach is the document management part of the client.

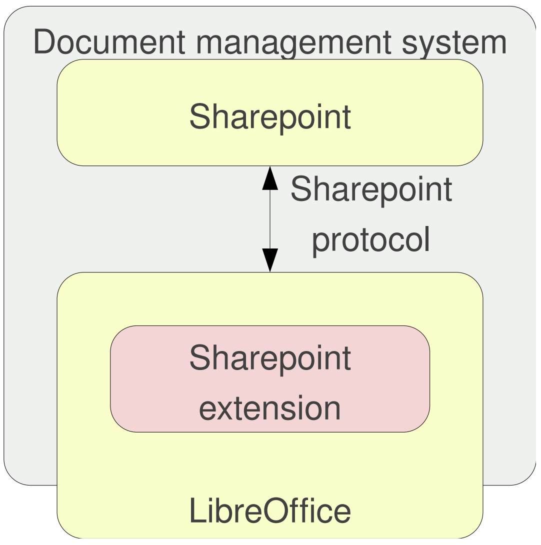


Figure 3.1. Architecture of the approach

Components of Figure 3.1 can be replaced, for example, Alfresco (with its SharePoint module) can be used in place of SharePoint, and OpenOffice.org can be used in place of LibreOffice.

The SharePoint server and the office productivity suite is already available, so at this stage the only remaining component is the SharePoint extension in LibreOffice that is missing, and our approach creates that.

Note that Alfresco already has an OpenOffice.org extension called OPAL [22]. That naturally uses Alfresco's native protocol to communicate with the server. The following major changes are needed to make it suit our needs:

- It's intended to be used with OpenOffice.org 3.1 – the last stable version of OpenOffice.org is 3.3, and it won't work with this version. The oldest stable release of LibreOffice is based on OpenOffice.org 3.3 as well, so porting the extension to 3.3 is essential to make it work at all with LibreOffice.
- It works by installing an additional module on the Alfresco server, so the business logic is minimal in the extension. My approach is to communicate without any server-side component installations required.
- The SharePoint protocol supports more features than Alfresco, so the user interface has to be extended to cover the new features.
- Finally, the protocol used has to be changed: by using the SharePoint protocol, the extension can communicate with both SharePoint and Alfresco, covering a lot of today's enterprise document management server installations.

Now that we understand what component we want to create and how, it's essential to understand what workflow the document management clients take part in.

The workflow has the following steps:

1. Create workspace: It creates a document workspace, which is the top-level container for any document stored on the document management server.
2. Browse workspace: An existing workspace can be browsed, and a documents can be put to workspaces.
3. Delete workspace: It's possible to get rid of no longer needed workspaces.
4. Put document: After selecting a target folder (using browse), a document can be uploaded.
5. Get document: Existing documents can be downloaded for viewing or editing.

6. Remove document: Unneeded documents can be removed individually.
7. List versions: Every upload of a document may (depending on its type) create a new version of the document. We can retrieve the list of this change log.
8. View previous version: older instances of a document can be viewed read-only anytime.
9. Restore version: it's possible to revert all changes after a given version with this step.
10. Delete version: older versions can be deleted.

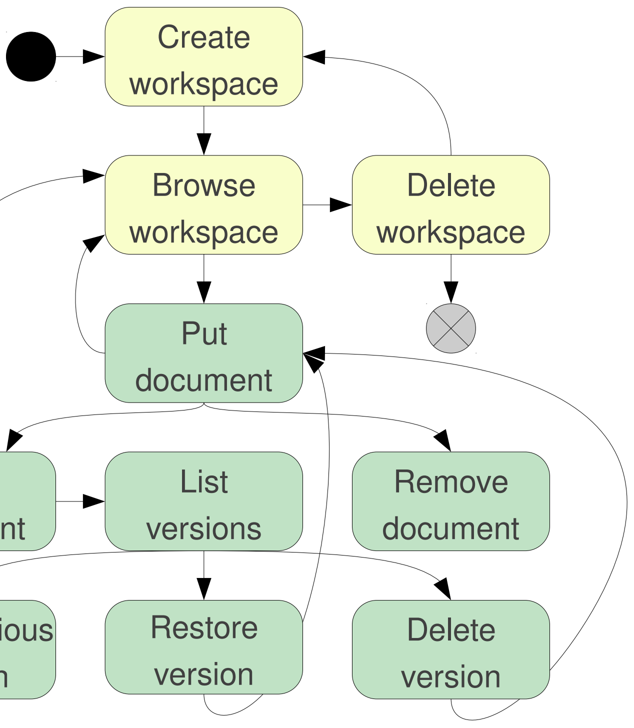


Figure 3.2. Workflow of the approach. Yellow: operations on workspaces, green: operation on documents.

Figure 3.2 shows what combinations of steps are allowed. An example:

1. A workspace is created.
2. The workspace is browsed.
3. A document is put to the workspace.
4. The workspace is browsed again.
5. The workspace gets deleted.

3.2 Workflows

Document and workflow server decoupling

Once the document-management client part of the system is in place, we propose a decoupled approach to connect the document management system and the workflow engine (Figure 3.3). We already saw what are the benefits of a system where each module can be replaced without altering the whole system, the motivation is the same here.

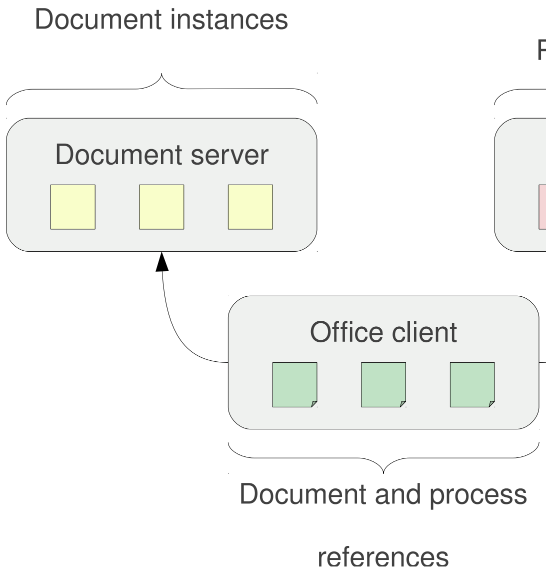


Figure 3.3. Architecture of the decoupled document based workflow approach

In our proposal the *document server and the workflow server is decoupled*, and the connection between the two is provided by the office client. Documents are stored on the document server, process instances are managed by the workflow engine, and the client will have references to both. Additionally, the workflow server will store references to documents, but such references are resolved by the client. As a result, the workflow server will not have to be altered in case the document server changes.

The system will have two operation modes:

- In case no workflow server is configured, or this is explicitly requested by the user, the extension will still connect to the document server only.
- Otherwise, the extension will query the tasks of the user in question from the workflow server, and that will be the basis of all future work.

The possible states of the latter mode of operation is demonstrated in Figure 3.4.

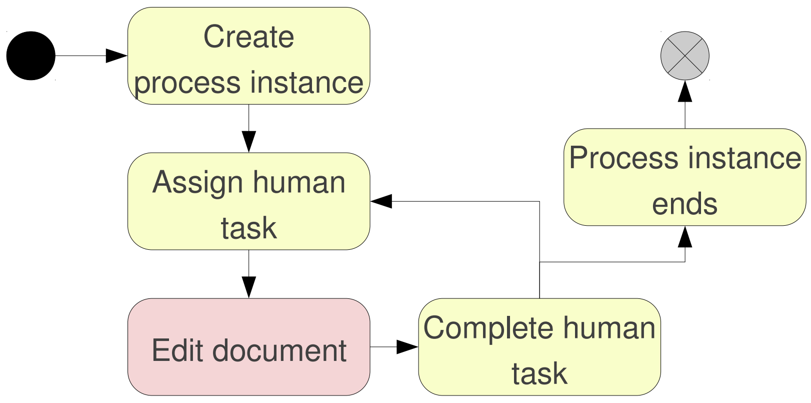


Figure 3.4. Control flow of the decoupled document-based workflow approach

The remaining part of this chapter describes each key feature we plan to design later.

Starting process instances

A process instance can be started externally or internally:

- If it is started externally, then the initiator of the process instance connects to the workflow console server using the console UI or using some other external application. After authentication, the user starts the process instance by manually specifying the reference (URL) of the associated document.
- A more integrated method is to start the workflow from the office client. First the associated document has to be opened, then the office client can be told to start a new process instance, once its type (a process definition) is selected from the process definition list.

Incremental snapshots

Once a process instance is started, its execution will be quickly blocked by waiting for a human task to be executed. The simplest case is when this human task is directly assigned to a user in the process definition. Given that each process instance has an associated document, we can talk about assigned documents. If a workflow server is configured, the file picker (the dialog that shows up when the user wants to open or save documents) will list these assigned documents for the user, providing them as documents in a virtual folder.

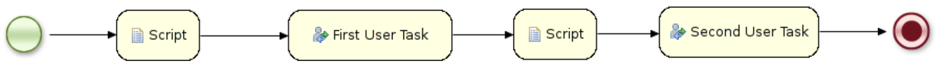


Figure 3.5. Workflow with two statically assigned human tasks

Now in case the workflow has two human tasks and statically assigned users (see Figure 3.5), the extension will offer completing the task when saving the document. *Incremental snapshots* is a feature that allows the user to choose if the task completion is wished or not, so the document can be incrementally saved to the document server as the task is being completed, and once it is done, the complete task operation is performed on the workflow server as well – without leaving the office environment for a second. Behind the scenes, once the last human task is completed as well, the process instance simply ends.

Group assignment

A natural extension of naming the assigned users in the process definition is to think about roles or groups, so the exact user performing the task can be determined later, at runtime. When the user reaches the file picker, a button can navigate to another virtual folder, called group tasks.

A new operation in this case will be *claiming a task*, which means selecting a document from the *group tasks* folder and moving it to the *personal tasks* folder. Once the task is a personal one, the user can proceed as before by editing the document, then completing the task.

The claim of a task can be undone: group tasks can be *released* – which means moving from the *group tasks* folder to the *personal tasks* one so other users can claim the task again.

Document masking

Once the currently opened document has associated workflow information, the editor can map parts of the document to parts of the process definition. Using this knowledge, the editor can limit write access to the relevant subset of the document, based on the current task instance associated to the document.

Decisions

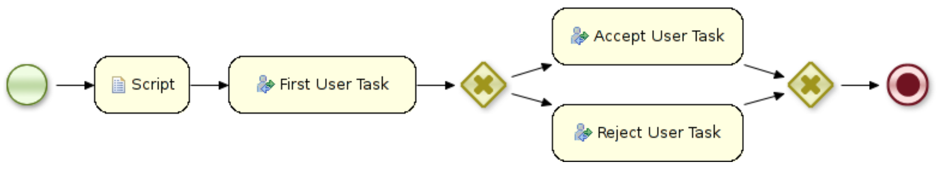


Figure 3.6. Workflow with a diverging and a converging gateway

Figure 3.6 shows a process definition containing a business decision. The problem is that in this case the *First User Task* can't be completed without arguments: a decision has to be made to determine which transition to fire after task completion.

The extension will query the list of possible choices from the workflow server and the user will be able to make the decision when saving the document.

Audit log

A benefit of using a workflow engine is the ability to query information about completed process instances. The usual question are:

- When did an action happen?
- Who performed the action?
- Where did it happen?
- What was the outcome of the action?

The extension will be able to show details of completed process, task and node instances, answering these questions.

In this chapter, we have shown what component of the document-based workflow system we plan to implement. Chapter 4 will describe how the implementation is designed to happen.

Chapter 4

Design

4.1 Document management

4.1.1 The SharePoint library

The Java implementation is split into two parts, the generic SharePoint library and the user interface, which is a UNO component. The previous lives under the *hu.ulx.lpsp.sharepoint* namespace, the other does so under *hu.ulx.lpsp.comp*.

The UML package diagram on Figure 4.1 shows these packages with their dependencies.

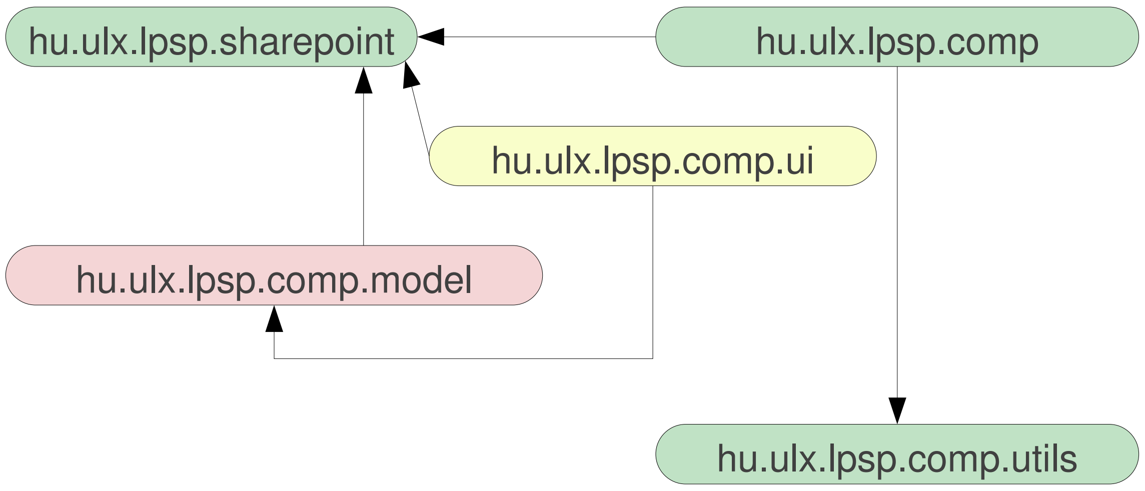


Figure 4.1. Packages of the document management extension

At the time of writing, there is no ready to use Java library to communicate with a SharePoint server. Because of this, we decided to separate the SharePoint protocol implementation from the LibreOffice-specific part of the extension to make it reusable in other projects. It provides the following classes:

- *SPHandler*: Handles requests from a frontend. Unless a feature has a specific class, it is implemented here.
- *FileOpenParser*: Parses the result from the *list directory of a workspace* request.
- *FileOpenRootParser*: Parses the result from the *list workspaces* request.
- *HandlerTest*: Automatically tests all features on a given server using JUnit.
- *LastModParser*: Parses the result from the *get last modification date* request.
- *Messages*: Provides localization for the library error messages.
- *PacketParser*: Extracts the error message from a Vermeer RPC [23] response packet.
- *Version*: Handles document versions.

We paid attention in our solution not to require additional server-side component installation, the SharePoint library can communicate with a standard SharePoint server without any server modifications.

4.1.2 The UNO interfaces

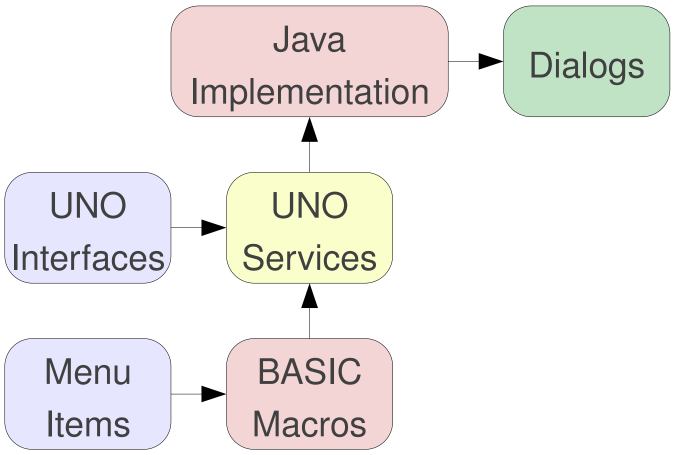


Figure 4.2. Technologies used in the design of the extension

The UNO interfaces are inspired by the already referenced OPAL extension. Where it made sense, we reused code from there, and changed it to work with Sharepoint.

It's a UNO convention that interfaces start with a capital X. The following **UNO interfaces** (see Figure 4.2) are provided by the extension:

- *XSharepointFilePicker*: an open/save as file picker.

- *XSharepointVersions*: a versions dialog.
- *XAuthenticationManager*: a handler for different authentication mechanisms.
- *XConnection*: a connection to a SharePoint server.
- *XSharepointDocument*: a document received from the SharePoint server.
- *XSharepointDocumentManager*: a storage for XSharepointDocument instances.

The extension provides a single exception type – *SharepointException* – when it throws errors.

It also provides a single enumeration type – *FileTypes* – to declare the list of LibreOffice applications it handles. (Its current value is: Writer, Calc, Impress and Draw.)

UNO services are interfaces containing static methods only. The following **UNO services** (see Figure 4.2) are provided by the extension:

- *Connection*: implementation for XConnection.
- *SharepointDocument*: factory for XSharepointDocument.
- *theAuthenticationManager*: singleton for XAuthenticationManager.
- *theSharepointDocumentManager*: singleton for XSharepointDocumentManager.
- *SharepointFilePicker*: implementation for XSharepointFilePicker.
- *SharepointVersions*: implementation for XSharepointVersions.

4.1.3 User interface

The user interface lives in the *hu.ulx.lpsp.comp* package. The classes of the user interface are organized into 4 Java packages:

- *hu.ulx.lpsp.comp.model*: a model for the folder and document structure, used by the file pickers (*open* and *save as* dialog windows).
- *hu.ulx.lpsp.comp.ui*: contains the dialog classes.
- *hu.ulx.lpsp.comp.util*: miscellaneous utility classes for UNO and localization.
- *hu.ulx.lpsp.comp*: classes implementing the rest of UNO services: authentication manager, document manager, document, connection, etc.

The user interface dialogs are located in the *Document repository* menu item, which is registered by the *Addons.xcu* configuration file, a standard configuration file of LibreOffice extensions.

Menu items

The following **menu items** (see Figure 4.2) are designed:

- Connection: force connecting to another document management server, even if the user is already connected.
- Open: downloads an existing document from the server.
- Close: discards the local copy of a downloaded document.
- Save: save the document model and upload the saved local copy to the server.
- Save as: upload the local document to the server using a new name.
- Versions: management of document versions.

The menu items call BASIC macros, which invoke UNO services. Those services are finally implemented in the Java user interface.

Macros

The macros are split into two packages: the ones directly called by the menu items, and the other utility macros.

The menu items call the following **macros** (see Figure 4.2) in the *LPSP.menu* package:

- openFile: shows the File Open dialog.
- openVersion: shows the dialog listing versions of the document.
- saveFile: saves the document via the document manager.
- saveAsFile: shows the Save As dialog.
- closeFile: closes the current document via the document manager, the already registered close listener will handle the cleanup of the local document copy.

The following ones are the utility macros (*LPSP.utils* package):

- getCurrentSharepointDocument: gets the document model of the current LibreOffice component (Writer document, Calc spreadsheet, etc.) from the document manager.
- getCurrentConnection: gets the current connection from the authentication manager.
- getConnection: shows the Connection dialog.

- getDocumentManager: returns the document manager singleton.
- getAuthenticationManager: returns the document manager singleton.

The BASIC macros invoke the following UNO services:

- Connection: calls *theAuthenticationManager::execute()*.
- Open: calls *SharepointFilePicker::execute()* with *IsOpen = true*.
- Close: calls *theSharepointDocumentManager::getSharepointDocument()*, then the *close* method on the returned result.
- Save: calls *theSharepointDocumentManager::getSharepointDocument()*, then the *save* method on the returned result.
- Save as: calls *SharepointFilePicker::execute()* with *IsOpen = false*.
- Versions: calls *SharepointVersions::execute()*

Java implementation

At this point, we know the user interface entry points in our Java implementation. As mentioned above, Java bytecode for the UNO services is automatically generated (by *javamaker*). The next task is to let the stub know where is its implementation. This is handled by the *RegistrationHandler* class. The list of implementation classes implementing a UNO service is in the *RegistrationHandler.classes* file. Each implementation class specifies what service it implements, then the registration handler collects this information and sends it to UNO, when asked. Finally, the *RegistrationClassName:* header in the *META-INF/MANIFEST.MF* file of the Jar package defines the registration handler class name.

As a result, the following entry points are designed in the **Java implementation** (see Figure 4.2):

- Connection: *AuthenticationManagerImpl.execute()*
- Open: *SharepointFilePickerImpl.execute()*
- Close: *SharepointDocumentManagerImpl.getSharepointDocument()*, then *SharepointDocumentImpl.close()*
- Save: *SharepointDocumentManagerImpl.getSharepointDocument()*, then *SharepointDocumentImpl.save()*
- Save as: *SharepointFilePickerImpl.execute()*

- Versions: *SharepointVersionsImpl.execute()*

These classes contain all the business logic and they call the SharePoint library for communication. The dialog windows are separated from UNO interfaces, they are always implemented in separate classes. Given that there are common tasks for all of our user interface dialogs, there is a common ancestor for all of them, called *AbstractDialog*.

Dialog classes

The following **dialog classes** (see Figure 4.2) are included in the *ui* package:

- Connection: *ConnectionDialog* (connect dialog),
  ConfigServerDialog (server list),
  ServerDialog (settings for an individual target)
- Open, save as: *FilePickerDialog*
- Close: None.
- Save: *CommentVersionDialog*
- Versions: *VersionsDialog*

For a UML class diagram showing these classes, see Figure 4.3.

4.2 Workflows

4.2.1 The workflow library

Similar to the document management support part, the Java implementation of workflow support is split into two parts as well: a generic jBPM client library and the workflow user interface. The jBPM client is designed under the *hu.ulx.lpsp.workflow* namespace, the second is under the *hu.ulx.lpsp.comp* one.

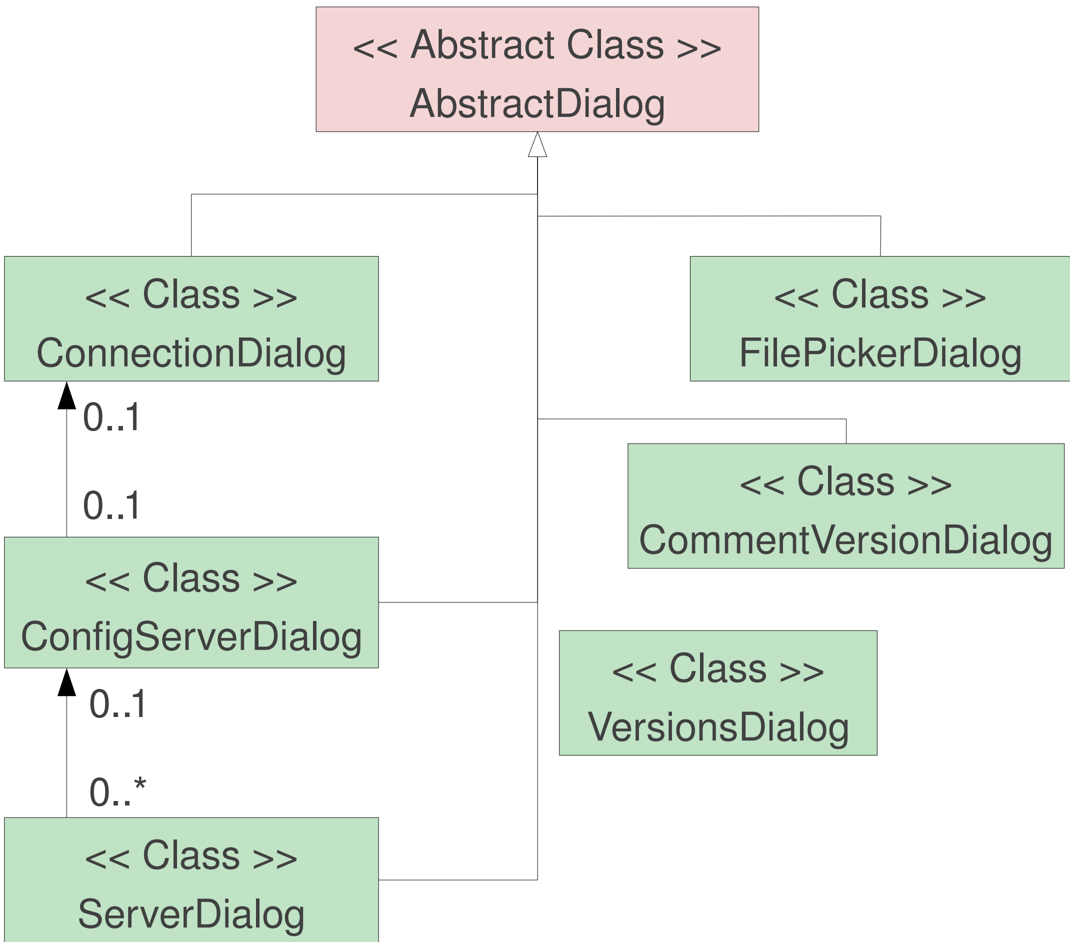


Figure 4.3. Classes of the user interface

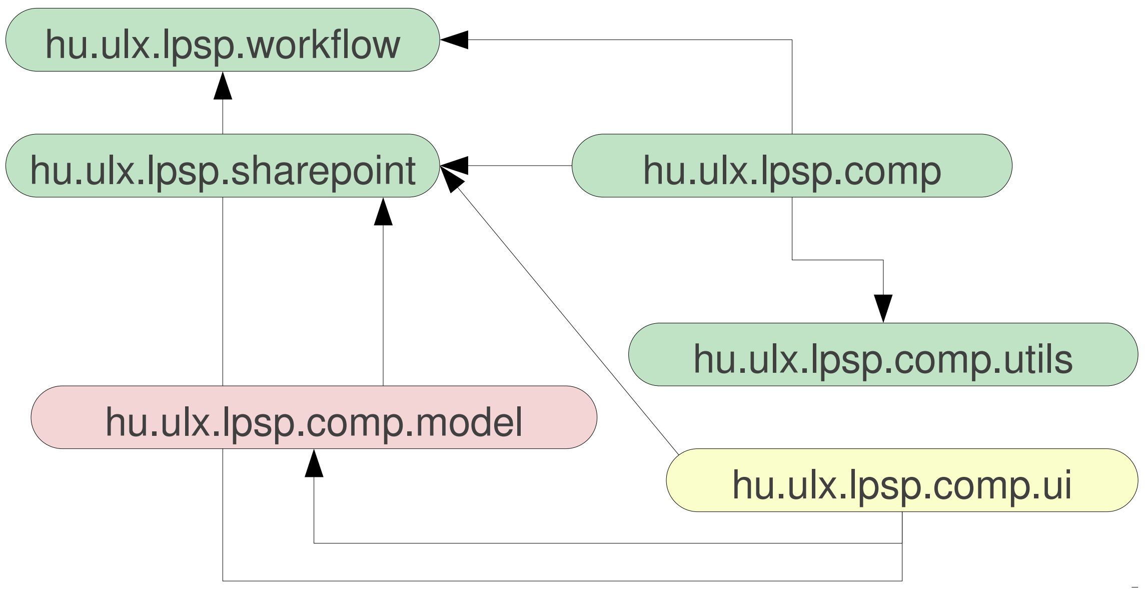


Figure 4.4. Packages of the workflow-enabled SharePoint extension

Figure 4.4 shows the updated UML package diagram of our solution. The recently introduced ***hu.ulx.lpsp.workflow*** package can be divided into three parts:

- controller classes
- entity classes
- parser classes

Controller classes

Given that jBPM provides a REST API to control the workflow engine, the client library can be quite simple. Its business logic is implemented in a single class, called *WFHandler*. It should provide the following features:

- authentication handling
- wrappers for the REST method calls used
- mapping between JSON data and entity classes
- parsing task data from HTML forms

Note that jBPM itself is stateless, but the client library mimics a stateful connection to avoid asking for connection parameters before all operations.

Parser classes

Whenever a REST method call returns from jBPM, the result can one of the following formats:

- HTML, when getting a task form
- XML, when asked for process instance data
- JSON, in any other case

There are multiple problems with arbitrary HTML in our case:

- jBPM expects an embedded HTML engine in the client, which does not exist in our SWING-based Java user interface
- once the form is submitted, the server would instruct the user to close the window, which would be inconsistent with our "the window is closed when no more input should be provided" approach
- in our case the form should be extended to ask for document-management details as well

When using HTML, there are several ways to express business decisions. Some example:

- using separate buttons for each decision
- using the HTML `<select>` tag to list choices
- using radio buttons
- some AJAX method

It is obviously impossible to support all these methods, so we declared a second constraint here: decisions should be described using submit buttons.

As a result, the *FormParser* class can simply parse submit tag name-value pairs from the form to detect multiple choices, and the list of values is sent to the user interface.

Entity classes

Whenever a JSON data structure is returned by a REST method call, the client library provides the results as an entity class. A convention we follow here is that every type has two entity classes:

- *TypeRef*: a reference to the instance of the given type itself

- *TypeRefWrapper*: a class for the collection of instance references of the given type

This is handy because this way we don't have to decide what Java interface (*java.util.List* could be a candidate) can cover all the details the server provides about an instance list.

The following types are handled by the client:

- *Process*: is a process definition
- *ProcessInstance*: is a process instance
- *NodeInstance*: is a node instance
- *Task*: a task instance

The type names are in sync with the REST API, which is more helpful than simply trying to be consistent with ourselves – and store the result of a task query in a task instance type.

4.2.2 jBPM and the BPM console server

The only functionality we need that is not provided by the BPM console server REST API is the audit log. The jBPM itself logs all the necessary information in its SQL backend, so we need the following steps to export this knowledge to the REST API:

- extend the interface of the integration layer in the BPM console server
- implement the extended interface in jBPM
- extend the REST API to use the extended interface

Given that jBPM and the BPM console are separate project, first we need an implementation in jBPM, then the new methods can be added to the integration layer interface in the BPM console.

Extending jBPM

We need three new features – the ability to get information about:

- historic process instances
- node instances of a process instance
- task instances of a process instance

In case of historic process instances, the following steps are needed:

- a new *getHistoricProcessInstances* method is introduced in the *ProcessManagement* interface implementation, calling
- a new *getInactiveProcessInstanceLogsByProcessId* method, to be introduced in the *CommandDelegate* class of the integration implementation package, calling
- a new *findInactiveProcessInstances* method, to be introduced in the *ProcessInstanceDbLog* class in the audit package of jBPM

This last method uses Hibernate to get the necessary information from the SQL database. The naming of the new methods is chosen to be consistent with the existing API.

Once the process instances are accessible, we need node instances. The following new methods are introduced to achieve this:

- a new *getNodeInstances* method in the *ProcessManagement* interface implementation, calling
- a new *getNodeInstanceLogsByProcessInstanceId* method, to be introduced in the *CommandDelegate* class

Modification of the *ProcessInstanceDbLog* class is not necessary, support for querying node instances is already implemented there. However, the *ProcessManagement* interface implementation should turn the Java object to a JSON serialization, which is handled by a new *nodeInstance()* method in the *Transform* class of the integration implementation package.

Finally, getting information about task instances of a process instance is a bit more complex. The following changes are introduced:

- A new *getProcessInstanceTasks* method in the *TaskManagement* interface implementation. This will instantiate a new *BlockingGetTasksResponseHandler* class, implementing a new *GetTasksResponseHandler* interface. Then it will call:
- A new *getTasksByProcessInstanceId* method in the *TaskClient* class. This will use two new constants (one is *QueryTasksByProcessInstanceId*, the other is *QueryTasksByProcessInstanceIdResponse*), provided by the *CommandName* class. This new method will call:
- the *TaskServerHandler* class, which should be extended to handle the *QueryTasksByProcessInstanceId* token, which will call:
- A new *getTasksByProcessInstanceId* method in the *TaskServiceSession* class. This will invoke a new stored procedure, named *TasksByProcessInstanceId*. Finally it will call:

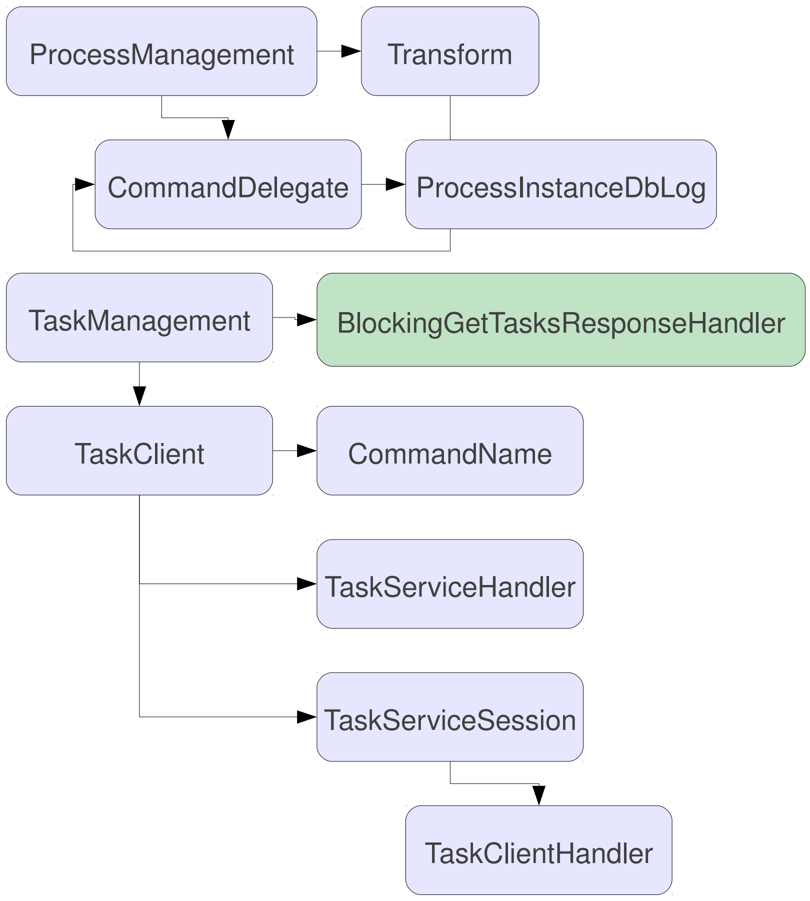


Figure 4.5. Audit log provider classes of jBPM. Blue classes are modified, the green class is a new one.

- the *TaskClientHandler* class, handling the *QueryTasksByProcessInstanceIdResponse* token, calling back the original *BlockingGetTasksResponseHandler* implementation.

The interaction between these classes can be seen on Figure 4.5.

Extending the BPM console server

Once the implementation of the integration layer in jBPM is in place, we can export these features in the REST API, using the following paths:

- `process/definition/history/{id}/instances`: process instances
- `process/history/{processInstanceId}/nodes`: node instances
- `tasks/history/{processInstanceId}`: task instances

When a query about a process instance is received, then:

- A new *getCompletedInstances* method in the *ProcessMgmtFacade* class is invoked, calling:
- The new *getHistoricProcessInstances* method in the *ProcessManagement* interface of the integration layer.

Similarly, if a query about a node instance is received, then:

- A new *getCompletedNodes* method in the same *ProcessMgmtFacade* class is invoked, calling:
- The new *getNodeInstances* method in the above *ProcessManagement* interface.

In this case serialization to JSON has to be handled by the new *NodeInstanceRef* and *NodeInstanceRefWrapper* classes. Note that these private classes are independent of the ones having the same name in the client library, as only the JSON output is public, allowing loose coupling.

Finally, once a query about a task is received, then:

- A new *getTasksForProcessInstance* method in the *TaskListFacade* class is invoked, calling:
- The new *getProcessInstanceTasks* method in the *TaskManagement* interface of the integration layer.

4.2.3 User interface

Workflow support affects the user interface in the following areas:

- menu items
- BASIC macros
- UNO interfaces and services
- Java Implementation classes
- Dialog classes

The rest of this chapter details these modifications.

Menu items

The following new menu items are introduced for workflow integration:

- *Start workflow*: starts a new process instance associated with the current document.
- *Workflow audit log*: shows a window with information about historic process instances.

BASIC macros

These menu items call the following BASIC macros in the *LPSP.menu* script library:

- *openWorkflow()* is called when the workflow definition list is opened.
- *workflowProcesses()* is called when the process instance list is shown.

UNO interfaces and services

To allow the macros to invoke Java methods, first we need a UNO specification. Two new interfaces are introduced:

- *XJbpmWorkflow*: invoke the Java library to start a new process instance.
- *XJbpmWorkflowProcesses*: invoke the Java library to show historic process instances.

Given that it's not possible to instantiate interfaces, new services are defined to implement these interfaces:

- *JbpmWorkflow*: implements *XJbpmWorkflow*

- *JbpmWorkflowProcesses*: implements *XJbpmWorkflowProcesses*

With this, the basic macros can call the following UNO services:

- Start workflow: calls *JbpmWorkflow::execute()*.
- Workflow audit log: calls *JbpmWorkflowProcesses::execute()*.

Java implementation classes

The UNO services invoke the following methods in the Java implementation:

- Start workflow: *JbpmWorkflowImpl.execute()*
- Workflow audit log: *JbpmWorkflowProcessesImpl.execute()*

Dialog classes

Figure 4.6 shows the new dialog classes used by the workflow UNO service implementations.

To start a new workflow, a single new dialog – *StartWorkflowDialog* – is planned, that will show a list of process definitions available.

The audit log will have three new dialogs:

- *WorkflowProcessesDialog* is the initial window, showing the process instance list.
- *WorkflowNodesDialog* can show node instances of a process instance, opened from the process instance list.
- *WorkflowTasksDialog* displays task instances of a process instance, similarly opened by the process instance dialog.

Additionally here and there existing dialogs have to be extended with workflow-specific items:

- *CommentVersionDialog*: When saving, new controls are needed to control decisions and completion of workflow tasks.
- *FilePickerDialog*: While opening, the file picker has to be improved to allow task claiming/releasing, and task listing should also be handled here.
- *ConfigServerDialog*: The table showing properties of different connections should show workflow server URL.

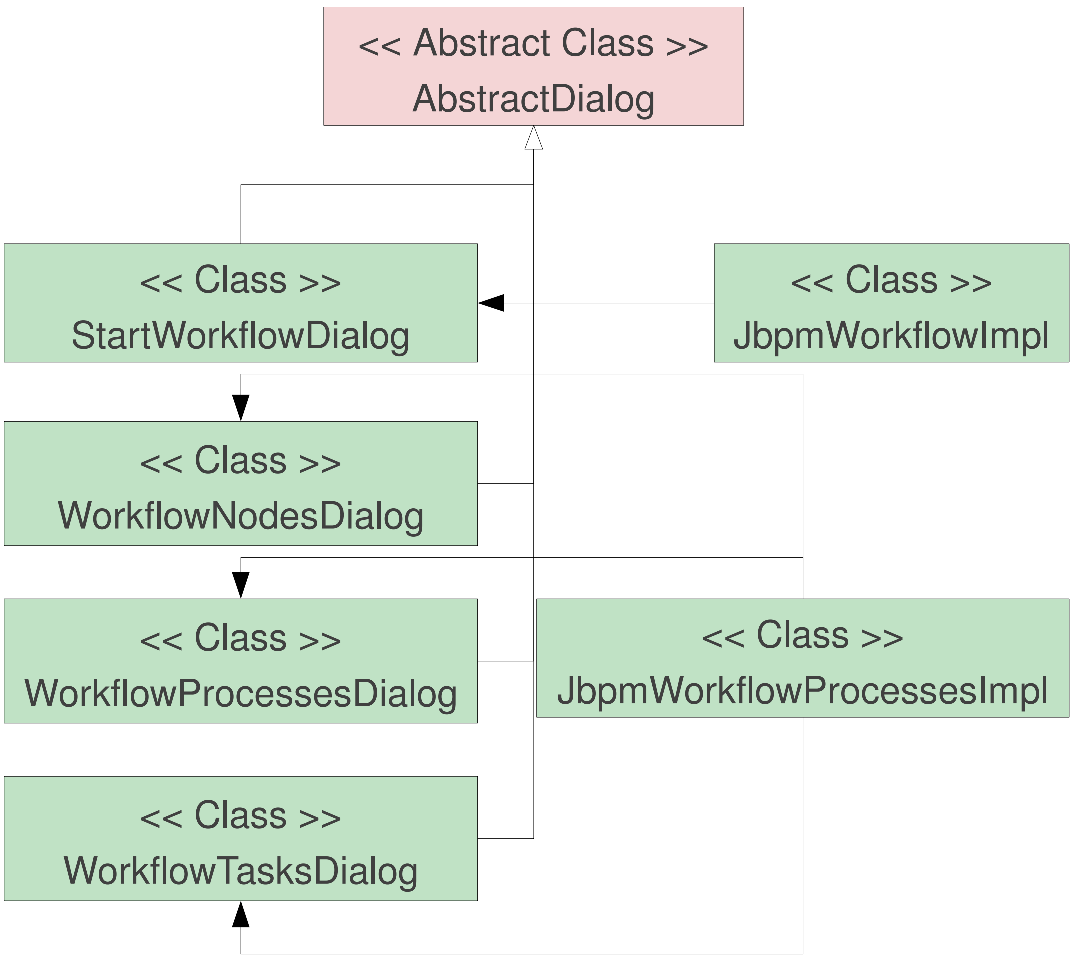


Figure 4.6. Classes of the workflow user interface

- *ServerDialog*: New controls are needed to be able to edit workflow URL of a server connection.

All modifications in the existing dialog classes should tolerate the lack of a workflow server, so document management integration can be used as a standalone feature as well.

Finally, a decision we made is we don't support all kind of process definitions, only the document-based ones. Our definition of being document-based is:

- the process definition has a string variable named *url*
- the process definition has a task form, where this URL can be specified on process start

Chapter 5

Implementation Details

5.1 Document management

The current section consists of three parts:

- The first one explains the prototype we wrote.
- The second one details implementation challenges when realizing the designed extension and using external libraries.
- The third one shows changes in the extension visible from a user's point of view, with screenshots.

5.1.1 The prototype

As mentioned already, the SharePoint protocol has a brief reference documentation on MSDN, but that is not enough to create an open-source implementation of the protocol. To solve that issue, we used two virtual machines:

- One running Microsoft SharePoint 2007.
- The other running Microsoft Office 2007.

Finally, we used Wireshark [24] to monitor the network traffic between the two virtual machines to reverse engineer the details of the protocol missing from the reference documentation.

Trying to reimplement the protocol directly in the LibreOffice extension would make development slow, partly because that would mean solving UNO interfacing issues and Java design decisions at the same time, partly because we needed a simple script to demonstrate how the protocol works, where Java may not be the best language to use. As a consequence, we decided to write a command-line prototype in Python, a popular scripting language, and once that was ready and worked, we ported the logic of the prototype to Java.

The prototype had commands (open, save, delete, etc.) to test each feature individually. This covered the functionality outlined in the *Background* chapter, except that folder / document listing is implicit here: the open and save-as operation invoked that, but it had no explicit command assigned.

The prototype also had two switches to test the implementation on both SharePoint and Alfresco automatically. This was critical, so that when we fixed something to work with Alfresco, we could quickly test that we did not break anything in SharePoint.

5.1.2 The SharePoint protocol

This subsection gives a brief overview of the SharePoint protocol; for the exact details, see the source code of our solution. Note that this analysis may not be fully accurate, it is only the understanding of the author, and was good enough to serve as a base of the implementation.

Definitions:

- workspace: a top-level directory, contains folders
- folder: a non-top-level directory, contains folders and documents
- document: several versions of the same file, identified by version numbers
- checkout, checkin, uncheckout: acquiring and releasing a lock on the document

The following workspace management operations are supported by the SharePoint library:

1. login: simply tries to access `/_vti_inf.html` on the server with the provided credentials
2. create workspace: SOAP request to `/_vti_bin/dws.asmx` in the site root
   - method name: `CreateDws`
   - parameters: `title` (name of the workspace)
   - returns: the HTTP status code (200 means success) can be used to check if the operation was successful
3. list workspaces: RPC request to `/_vti_bin/_vti_aut/author.dll` in the site root
   - method name: `list documents`

- parameters: none
- returns: HTML page, a single enumeration provides the URL of the workspaces

4. delete workspace: SOAP request to `/_vti_bin/dws.asmx` in the workspace root
    - method name: `DeleteDws`
    - parameters: none
    - returns: the HTTP status code (200 means success) can be used to check if the operation was successful

Folder management:

1. create folder: SOAP request to `/_vti_bin/dws.asmx` in the site root
    - method name: `CreateFolder`
    - parameters: `url` (url of the folder)
    - returns: the HTTP status code (200 means success) can be used to check if the operation was successful
2. list folders: HTTP GET to `/_vti_bin/owssvr.dll` in the workspace root
    - parameters: `location` (URL of the folder), `FileDialogFilterValue` (filter to apply to the result on server side, for example `*.*`)
    - returns: HTML page, containing a table where every row is a folder or a document and each column contains properties of the item. The property names can be configured in runtime on the server.
3. delete folder: SOAP request to `/_vti_bin/dws.asmx` in the site root
    - method name: `DeleteDws`
    - parameters: `url` (url of the folder)
    - returns: the HTTP status code (200 means success) can be used to check if the operation was successful

Document management:

1. open: no dedicated method, HTTP GET should be used with the URL's returned by *list folders*
2. save: RPC request to `/_vti_bin/_vti_aut/author.dll` in the workspace root

- method name: `put document` (yes, RPC method names can contain spaces)
- parameters: `service_name` (name of the workspace), `document` (structure containing the document name and last modification date), `comment` (this option should be provided, but its value is ignored)
- payload: everything after two newline characters is considered as the document data
- returns: an RPC response packet, containing a "successfully put document" string with a `msg=` or `message=` prefix

3. remove documents: RPC request to `/_vti_bin/_vti_aut/author.dll` in the workspace root
   - method name: `remove documents`
   - parameters: `url_list` (list of URL's to remove)
   - returns: an RPC response packet, containing a "successfully put document" string with a `msg=` or `message=` prefix
4. get document metadata: RPC request to `/_vti_bin/_vti_aut/author.dll` in the workspace root
   - method name: `getDocsMetaInfo`
   - parameters: `url_list` (list of URL's to query)
   - returns: several HTML enumerations, one for each URL

Lock management:

1. checkout: SOAP request to `/_vti_bin/lists.asmx` in the workspace root
   - method name: `CheckOutFile`
   - parameters: `pageUrl` (URL of the document to check out), `lastmodified` (last modification date of the document, in case the document is already opened but not checked out)
   - returns: `CheckOutFileResult`, containing a boolean value
2. checkin: SOAP request to `/_vti_bin/lists.asmx` in the workspace root
   - method name: `CheckInFile`
   - parameters: `pageUrl` (URL of the document to check in), `comment` (the "commit message") and `CheckinType` (0 = `MinorCheckIn`, 1 = `MajorCheckIn`, and 2 = `OverwriteCheckIn`)

- returns: `CheckInFileResult`, containing a boolean value

3. undo checkout: SOAP request to `/_vti_bin/lists.asmx` in the workspace root
   - method name: `UndoCheckOut`
   - parameters: `pageUrl` (URL of the document to uncheckout)
   - returns: `UndoCheckOutResponse`, containing a boolean value

Version management:

1. create version: no dedicated method, a *save* or *checkin* can create a new version
2. list versions: SOAP request to `/_vti_bin/versions.asmx` in the workspace root
   - method name: `GetVersions`
   - parameters: `fileName` (URL of the document)
   - returns: a list of `result` elements – each contains "version", "created", "createdBy", "size", "comments" and "url" fields
3. restore version: SOAP request to `/_vti_bin/versions.asmx` in the workspace root
   - method name: `RestoreVersion`
   - parameters: `fileName` (URL of the document to restore), `fileVersion` (old version number)
   - returns: a list of `soap:Fault` elements – if it is empty, it means success
4. delete version: SOAP request to `/_vti_bin/versions.asmx` in the workspace root
   - method name: `DeleteVersion`
   - parameters: `fileName` (URL of the document), `fileVersion` (version number)
   - returns: a list of `soap:Fault` elements – if it is empty, it means success

In all SOAP requests an additional `X-Office-Version` header is necessary, the value of this header is 12.0.6514 for Office 2007.

In RPC requests, the two following headers are necessary:

```
Content-Type: application/x-vermeer-urlencoded
X-Vermeer-Content-Type: application/x-vermeer-urlencoded
```

Once the details of the protocol – at least the parts required for our use-cases – were clear, we could begin writing the LibreOffice extension.

5.1.3 External libraries

It was obvious that doing HTTP communication with NTLM authentication is an already solved problem, but we needed to decide which library to use. Given that OPAL already used Apache *commons-httpclient* [25] 3.x for HTTP communication, that was an option. Unfortunately that version does not support NTLM, so we used Apache *httpcomponents* [26], which is the successor of the previous library (starting with version 4.x). The two version series have different APIs, so it was possible to temporarily use both in parallel, and migrating code incrementally.

Once we had *httpcomponents*, we still needed to write detection code that decided what to request from *httpcomponents*: Basic or NTLM authentication.

An other interesting issue was to parse the response received after sending Vermeer RPC requests to the server. The result is valid HTML, but not XHTML. For example, part of the response is:

```
<html><head><title>vermeer RPC packet</title></head>
<body>
...
<li>vti_timecreated
<li>TR|27 Feb 2011 19:07:25 +0000
<li>vti_timelastmodified
<li>TR|11 Mar 2011 16:39:35 +0000
<li>vti_timelastwritten
<li>TR|11 Mar 2011 16:39:35 +0000
...
</body>
</html>
```

That means a simple XML parser was not enough to extract the needed values from this response. To solve this issue, we used TagSoup [27], which is a SAX parser, accepting plain old HTML input.

5.1.4 User interface

Once the Sharpeoint Library was ready, we updated the user interface to use the SharePoint library for communication. We also had to extend the dialogs to allow a few more features. Namely:

- Create and delete document workspaces.
- Delete documents.
- Delete and restore versions.
- When saving a document, allow: minor change with a comment and overwrite of a previous version.

For example, creating a new document workspace is implemented as can be seen on Figure 5.1.

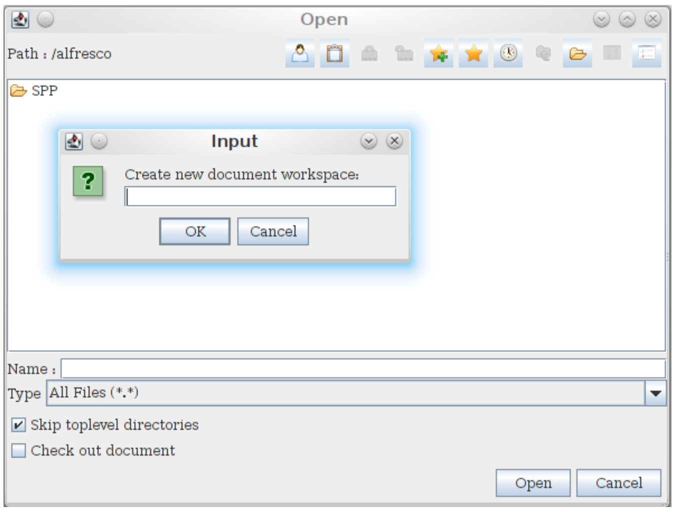
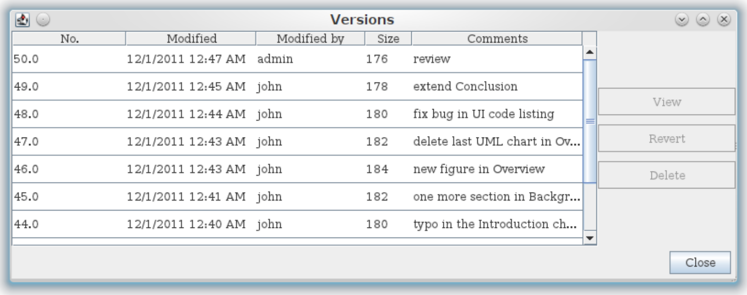


Figure 5.1. Implementation of creating a document workspace

Earlier only viewing older versions was possible; the new *Versions* dialog now allows a user to delete and restore versions as well (as detailed above in Table 2.1, deleting versions does not work with Alfresco, due to the shortcomings of their SharePoint implementation), see Figure 5.2.



No.	Modified	Modified by	Size	Comments
50.0	12/1/2011 12:47 AM	admin	176	review
49.0	12/1/2011 12:45 AM	john	178	extend Conclusion
48.0	12/1/2011 12:44 AM	john	180	fix bug in UI code listing
47.0	12/1/2011 12:43 AM	john	182	delete last UML chart in Ov...
46.0	12/1/2011 12:43 AM	john	184	new figure in Overview
45.0	12/1/2011 12:41 AM	john	182	one more section in Backgr...
44.0	12/1/2011 12:40 AM	john	180	typo in the Introduction ch...

Figure 5.2. Implementation of version restore and delete

5.1.5 Exception handling

The last challenge was the localization of error messages in BASIC. The language does not support any kind of localization. What it supports is the following technique (control flow shown on Figure 5.3):

```
Sub openVersion
...
        On Error GoTo errSpDoc
        oVersion.execute()
        Exit Sub

        errSpDoc:
                MsgBox(oVersion.ErrorReason)
                Exit Sub
End Sub
```

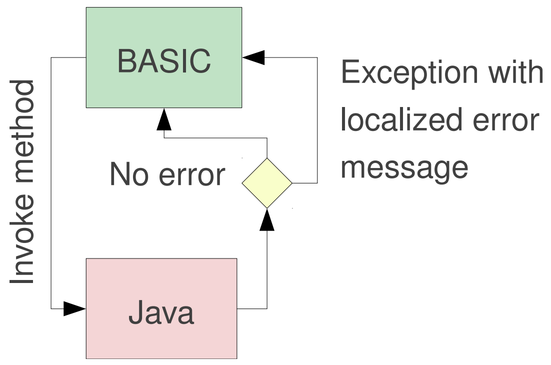


Figure 5.3. Control flow of localized error messages in BASIC, thrown by Java.

The *On Error* part can hide any kind of exception thrown by Java, and as long as the Java side calls *setErrorReason()* on the *SharepointVersionsImpl* object before throwing an exception, the user gets a friendly error message, which is even localized.

5.2 Workflows

The implementation of workflow integration had multiple interesting challenges:

- adding support for the authentication method jBPM uses
- handling data received in responses from jBPM
- error handling in multiple levels

5.2.1 Authentication

The jBPM uses form-based authentication. That means that in case the user is not yet logged in, he/she is redirected to a login form, where the user name and password should be submitted. The *httpcomponents* package provides a dedicated *UrlEncodedFormEntity* class for this purpose:

- Fields in the form should be collected in a *NameValuePair* list;

- Once the list is ready, the entity generator class can produce the necessary payload for the HTTP request, using the `application/x-www-form-urlencoded` MIME type [28].

There are two problems here:

- The user interface expects a stateful connection here, while HTTP is stateless.
- The password is sent without encryption.

The first problem can be solved multiple ways. First, the workflow handler can remember the credentials passed from the user interface, hiding the repeated login procedure from the user. Alternatively, in this case jBPM handles sessions, so receiving and sending back cookies avoids multiple logins as well.

The problem of clear-text passwords sent over the network can be solved at an application server level, in case HTTPS is used.

From a user experience point of view an additional problem is the confusing connection dialog that asks for multiple user names and passwords. Given that we are integrating the document management and the workflow system, we can expect that the same credentials are used by the same users to access these servers. As a result, the user interface asks for a single login and password, the Java implementation does two logins in practice, though. The user interface signals an error in case any of these logins failed.

5.2.2 JSON handling

Most responses received from the jBPM server are JavaScript Object Notation (JSON) strings. The interested reader can look into the specification [29], all we need to know about it here is that it can represent a set of Java objects, allowing references between them.

Once entity classes are available (as described in the *Design* chapter), the Gson library [30] can be used to parse those strings to entity class instances.

The benefits of using Gson over other implementations (like *org.json* [31]):

- it requires no pollution of the entity classes regarding JSON (i.e. no *toJson()* methods are necessary)
- when a default constructor is not available for an entity class, it supports the registration of instance creators for such types

It uses reflection to access fields, so the general object-oriented methodology where the fields themselves are private and public getter/setter methods are provided is possible to use.

Also, the parser is quite liberal about entity classes, in case a field is present in the JSON string but the associated Java class does not have such a field, it is simply ignored. That helps our extension to be compatible with future jBPM server versions.

A pitfall during the implementation was that the usage of generic types is mandatory when using references between the classes:

- In case a generalized collection is used as a field type, Gson can extract the type information of the referenced objects from the field type, and instantiate the correct class.
- Otherwise, there is no way to guess the type of the referenced instance, so a parse exception is thrown.

5.2.3 User interface

Private streams

The first challenge was to map the concepts of a user and the associated personal / group tasks to the file / folder metaphor, already known by users.

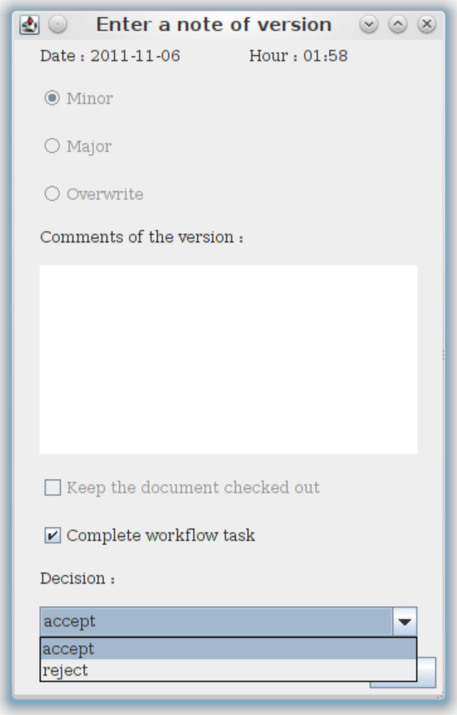
In LibreOffice, access to files is usually handled by a sequence of property values (key-value pairs), where usually the *URL* property is used to access the physical file. However in some cases there is no real URL, but the contents of the object are provided as a *Stream* property. An example is the framework module [32], where even a define (with a value of *private:stream*) is provided to be used as a fake URL for such streams. A similar constant is available to access factories with a *private:factory* URL. Using that URL e.g. in Writer, a new empty document is opened.

Extending this technique, the extension introduces the following identifiers in the *private:* namespace:

- *private:personaltasks* is used to access personal tasks
- *private:grouptasks* is used to access group tasks

The *WFHandler.urlopen()* method, which was originally designed to open regular URL's, can handle these special ones. During the opening operation it performs the following steps:

- It queries task instances of the given task type from the workflow server.
- Once a task instance is selected, it extracts the associated document URL from the associated process instance variables.
- Finally, the extension already knows how to open a document server URL.


Enter a note of version

Date : 2011-11-06 Hour : 01:58

( ) Minor

( ) Major

( ) Overwrite

Comments of the version :

[ ] Keep the document checked out

[✓] Complete workflow task

Decision :

accept

accept
reject

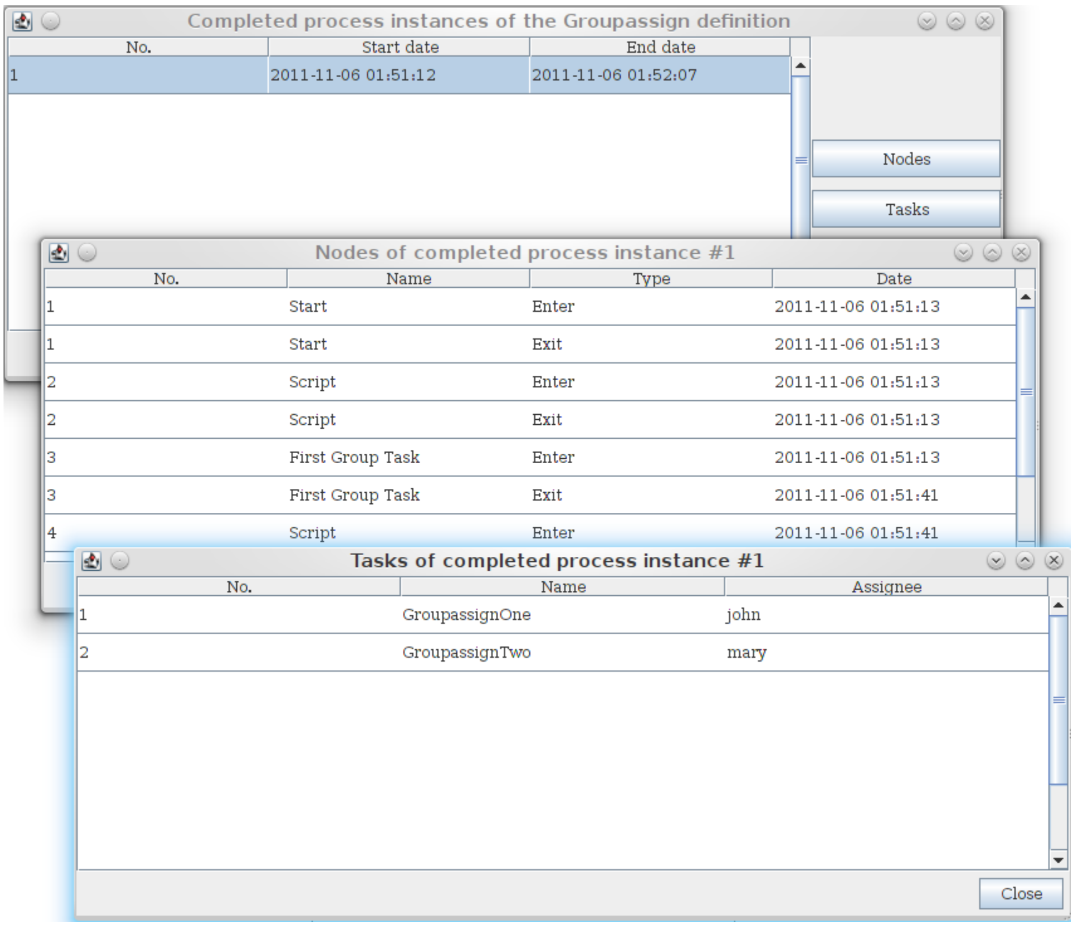
Figure 5.4. Implementation of making a business decision when completing a human task

During save, the *CommentVersionDialog* class invokes *WFHandler* to check if the document has a task associated. If so, the two controls at the bottom of the dialog are enabled to control the interaction with the workflow server during save (see Figure 5.4):

- if the workflow should be stepped
- the answer to the business decision, if available

Audit log

The second new menu item introduced by the workflow integration is the audit log. First a dialog asks the process type to audit (similar to the process definition selector when starting a new process instance), then the list of historic process instances appear.


Completed process instances of the Groupassign definition
Nodes
Tasks

No.	Start date	End date
1	2011-11-06 01:51:12	2011-11-06 01:52:07

Nodes of completed process instance #1

No.	Name	Type	Date
1	Start	Enter	2011-11-06 01:51:13
1	Start	Exit	2011-11-06 01:51:13
2	Script	Enter	2011-11-06 01:51:13
2	Script	Exit	2011-11-06 01:51:13
3	First Group Task	Enter	2011-11-06 01:51:13
3	First Group Task	Exit	2011-11-06 01:51:41
4	Script	Enter	2011-11-06 01:51:41

Tasks of completed process instance #1
Close

No.	Name	Assignee
1	GroupassignOne	john
2	GroupassignTwo	mary

Figure 5.5. Implementation of the audit log: process instances, node instances and users performing human tasks

Buttons next to this table are available to inspect node and task instances associated to the process. With this, we can answer all of our original questions:

- When did an action happen? Date columns of the process and node dialogs.
- Who performed the action? Assignee column of the task dialog.
- Where did it happen? Name column of the process and task dialogs.
- What was the outcome of the action? Name column of the node dialog.

5.2.4 Error handling

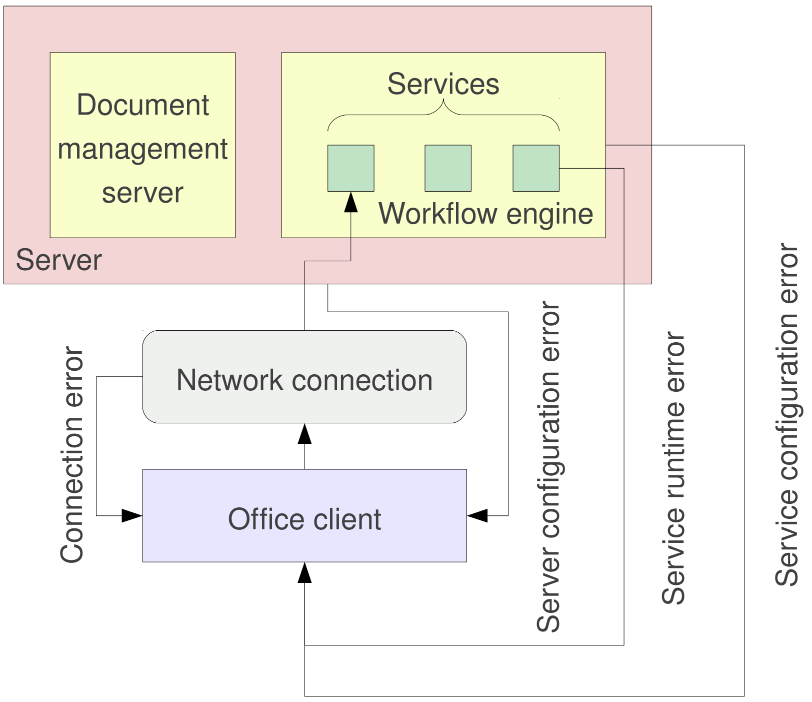


Figure 5.6. Different levels of error handling in the extension

There are multiple levels where error handling should be implemented (Figure 5.6):

- Accessing jBPM without a connection.
- Accessing jBPM with a connection not configured for workflows.
- Accessing a jBPM service which is not implemented by the server instance in question.
- Handling errors from existing services.

Accessing without a connection

Detecting this status is implemented in the BASIC macros:

- Macros call the *getCurrentConnection()* function get the connection reference.
- *getCurrentConnection()* gets the current connection from the authentication manager (which is always available, since it is a singleton).
- In case there is already a connection, it returns the reference. If there is no connection, it calls the *getConnection()* function, that shows the connection dialog.

In case the user cancels the connection dialog, the action is cancelled, this way the Java implementation can assume there is always a connection available.

Accessing without workflow support

The user always has the possibility to leave the workflow URL of a server configuration empty. In this case the extension does not try to connect, either.

Later during workflow-related operations the control flow always starts with the following steps:

- get a reference to the current connection
- call the *getWFHandler()* method of the connection, to check if workflow support is available

In case the second step fails, the user gets an error message explaining the issue.

Checking workflow services

The jBPM support listing available services, however that is more about serving as a user guide to show what can be done, rather than a page to be machine-parsed. The already mentioned TagSoup parser could be used to extract the necessary information, but the page is not part of any stable API, so that approach could easily break with the next jBPM version.

A better method is to properly handle all JSON parsing exceptions. This is possible because in case a service is not supported, a HTML error is returned by the server, which – of course – cannot be handled by the JSON parser.

Using this technique we can inform the user about exactly what required service is missing on the server.

Errors in existing services

A rather generic approach would be to check the HTTP code to detect errors, but it has two problems:

- The server returns HTTP 200 (*OK*) even in case the HTML title of the page is *HTTP 401*.
- Even if the code would not be 200, we can't distinguish between service errors and non-existing services, unless different error codes are used (e.g. 500 for service errors, 503 for unavailable services).
- One can also argue that HTTP error codes are to be used by the application server itself, not by servlets running on the server.

A better method is to check the content-type, and – in case it is HTML – the title of the page. If the content-type is unexpected, that will mean the service is not available, as described above. Otherwise, the title of the returned HTML page correctly contains the error code.

An example is when a login fails, in that case the *HTTP 401* error code is properly returned in the rendered HTML error page.

Chapter 6

Evaluation

6.1 Document management

In this section, we take Table 2.1 as a starting point and check what features are implemented and what are not.

The testing environment was built from a client machine, a SharePoint server and an Alfresco server, as can be seen on Figure 6.1.

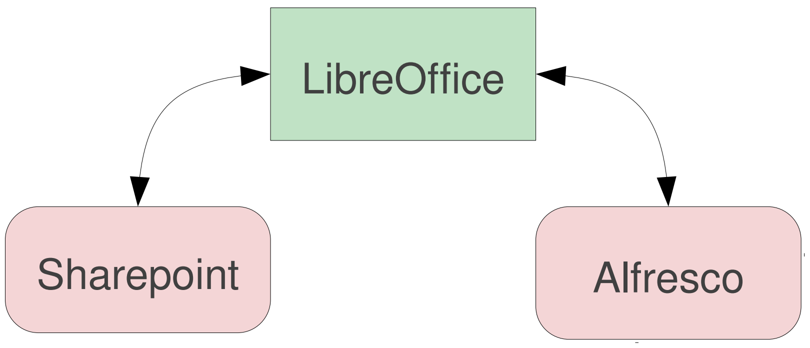


Figure 6.1. Architecture of the test system

The client machine had the following details:

- Frugalware Linux 1.5 (English and Hungarian locale)
- LibreOffice 3.4, LibreOffice 3.3, OpenOffice.org 3.3 and OpenOffice.org 3.2.1

Properties of the SharePoint machine:

- Microsoft Windows Windows Server 2003 R2 Enterprise (English)
- Microsoft SharePoint 2007 Enterprise

Finally the Alfresco machine:

- Fedora Linux 16
- Alfresco 3.4.d Community Edition

Regarding functionality, all items from the feature table are implemented in the extension. The following areas are missing from the table, and not supported at the moment:

- User management, including roles and permissions inside workspaces.
- Task management in workspaces.
- Link management in workspaces.

The following items are not in the feature table, but are available:

- Creating and deleting workspaces is supported by Microsoft Office, but nested folder structures can only be read. It turned out that the protocol allows creation, so the extension supports this.
- Namespaces of settings and classes are renamed, so parallel installation of our extension and OPAL is possible.

The extension is written in Java and BASIC, so it is meant to be portable. LibreOffice is available on Windows and OSX as well – so the extension should work there, but we only had time to test it on Windows.

Regarding localization, all user-visible strings are externalized to Java property files. During development we paid attention to English strings, then at the end as a demonstration we created the Hungarian translation as well. Adding new translations is easy.

Finally, the extension inherited some of the limitations of the original OPAL codebase:

- The GUI is not started in a separate thread in all cases, so in many cases the user interface is not updated in the background.
- Classes we did not touch still have French comments inside.

6.2 Workflows

Test architecture

The evaluation of workflow support is based on the use-cases of the workflow part of the *Overview of the Approach* chapter.

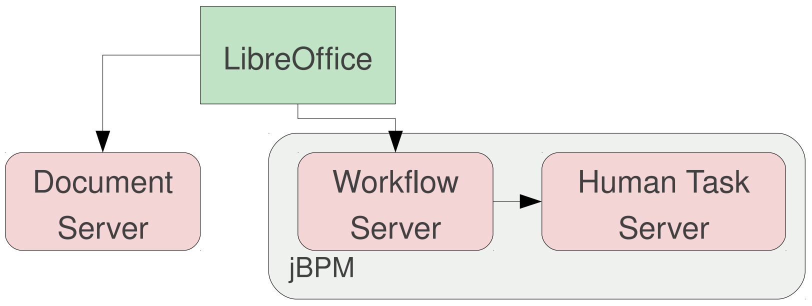


Figure 6.2. Architecture of the workflow-enabled test system

We designed a test architecture to evaluate these features, and Figure 6.2 shows the structure of this test environment. The test system has the following dependencies:

- The connection should be initiated from our LibreOffice extension when both the document server and the workflow engine is ready.
- The workflow server can be started before the Human Task server, but user connections can only be served when both are ready.
- Additionally, in case a custom storage backend is used for jBPM (for example MySQL instead of the default H2 backend), then that should be ready before any component of jBPM is started.

During our tests we used the same document servers and office suites than for the document-server-only tests. The jBPM instance was set up on the same physical machine where the office suite ran.

The workflow-specific tests were executed using the following component versions:

- jBPM (jbpm-bpmn2 and jbpm-human-task) 5.1.x: latest version at the time of writing (November 2011) with additional patches detailed earlier.
- BPM Console 2.1 with the patches.
- MySQL 5.5.14 without additional modifications.

The jBPM developers are looking forward to the modifications and are planning to review them once the thesis is ready.

Used test methods

Three different test methods were used:

- The existing document management unit tests were used to ensure that we do not break the document-management-only use cases (see Table 2.1).
- Separate command-line applications were used to test features of the workflow library. These are to be turned into unit tests.
- Due to the complex architecture, integration tests were performed manually.

The last method does not only mean it was continuously tested by the author: a closed set of users also tested it once the document management part was ready. They were already used to using a document server and office suite separately, and they all had the possibility not to use the extension if they thought it complicated their daily work. The general feedback was:

- The extension made their work easier: in most cases they did use the extension once it was installed.
- Even if they had to report bugs here and there, the extension never damaged data on the servers (e.g. by accidentally deleting a version of a document).

Limitations

The workflow support part has the following known limitations:

- Process definitions without a start form – where the associated document URL can be specified – are not supported.
- Free-form HTML is not allowed in task forms when making a business decision.
- The audit log features are not available with an official jBPM version.

The first two are design decisions, the last one will be resolved in the near future, as mentioned earlier.

However the document management integration also causes two limitations, which affects workflows as well:

- The SharePoint protocol does not allow listing files without an extension. This means when using the extension to start a workflow, such files can't be associated to the process instance.

- By choosing a decoupled approach, we also require an external method to be used for keeping user accounts on the document management and workflow servers in sync.

Here the first item is a known limitation, the second one is a design decision (see Chapter 4).

Chapter 7

Related Work

7.1 Document management

We already saw that there is no solution today that is open-source, requires no additional server-side installation and communicates with a Microsoft SharePoint server (see Table 7.1).

However, there are similar projects, which – even if they solve a sightly different problem – may provide excellent ideas to borrow.

OPAL

We already introduced OPAL, which is open-source, but:

- Requires server-side modifications.
- Works with an Alfresco server.
- Not really maintained (does not work with latest stable OpenOffice.org).

However, its user interface and concepts are quite similar to our solution.

LibreOffice CMIS

LibreOffice CMIS [33] is a LibreOffice extension: a Java based implementation of a Universal Content Provider for making any content in a CMIS repository usable from LibreOffice. The main problem with it is that SharePoint 2007 does not implement CMIS, so this does not solve our problem at the moment.

SharePoint Connector

The Oracle Connector for SharePoint Server [34] is a commercial OpenOffice.org extension, providing SharePoint support in OpenOffice.org. It has multiple problems:

- It is not free. Sadly it is part of Oracle Open Office Enterprise Edition, which is no longer available from the Oracle Store at the time of writing (November 2011).
- When we checked it earlier, it was only available on Windows 32-bit and it required a server-side component as well.

libcmis

The libcmis [35] library is a general purpose CMIS library, written in C++. LibreOffice has a Universal Content Provider built into its core, internally using libcmis.

It has two issues at the moment:

- its CMIS support is incomplete
- like LibreOffice CMIS, it does not support SharePoint 2007

The first problem is expected to be resolved in the long term, the second is not a priority for this project.

Drupal SharePoint module

An additional SharePoint module [36] for the Drupal content management system is also available. It solves a different problem, though:

- It can only read from a SharePoint server.
- It is written for a CMS, not an office suite.
- To achieve its goals, it was enough to use the SOAP interface of the SharePoint server, which is not enough for our purposes.

Feature	**OPAL**	**LibO CMIS**	**libcmis**	**SP Conn.**	**Drupal**
License	open-source	open-source	open-source	proprietary	open-source
Server component	yes	no	no	yes	no
SharePoint 2007 support	no	no	no	yes	partial

Table 7.1. Comparison of related document management systems

7.2 Workflows

Table 7.2 shows that there is no ready approach today for document-based workflow management, that

- supports arbitrary process definitions
- decouples document and workflow management
- is available under an open-source license

On the other hand, these are partly detailed in papers and implemented in some other projects, which we present here.

Related papers

Document-based workflows[1] is an active research topic today. Multiple interesting ideas are raised in recent papers.

A framework for document-driven workflow systems [37]:

- details why information and resource based workflows are also important, not just control based ones
- use case: user changes the document, change listeners intercepts changes, check constraints, then accept or reject them
- various complex features planned: split/merge of documents, different locking types
- proposed implementation using SQL triggers
- detailed comparison of control flow based versus document based approach

The proposed implementation is heavily storage-dependent, while our decision is to interoperate with existing storage solutions.

Mobility in the virtual office: a document-centric workflow approach [38]:

- introduces decentralized document-driven workflows
- changes to workflows travel with the documents
- decentralization is handled with a peer-to-peer architecture

[1] Also known as document-centric or document-driven workflows.

This paper proposes a decentralized architecture, our method is simpler, having a centralized design.

XDoC-WFMS – A Framework for Document Centric Workflow Management System [39]

- presents use cases of intiutive workflow and document management integration: newspaper editing, processing job applications
- proposed solution: documents have an embedded micro-agent, so the document itself will know where to go after a task is completed

The suggestion is to integrate executable code with documents, while our approach is to decouple the workflow engine and the document servers.

Access control in document-centric workflow systems – an agent-based approach [40]

- this approach is without decoupling as well
- the workflow object is proposed to communicate with the document object

This approach highlights the importance of access control, our extension simply threats it as an existing building block.

SharePoint Designer

The SharePoint Designer 2007 tool [41] supports designing process definitions, to be executed within the SharePoint document management server itself. It focuses on two features:

- triggers, executing an action when a document-related event occurs
- a few builtin process definitions (review, approval, collecting signatures)

Its problems for our purposes:

- decoupling of the document and workflow server
- document masking
- standard workflow format (such as BPMN)

Liferay

Liferay[2] [42] is an open-source content management system, focusing on the needs of enterprises. Regarding document management, it comes with a *Document Library* feature, which is similar to the one Sharepoint and Alfresco provides.

The Document Library publishes the contents via WebDAV, so basic file operations (open, save) are simple from an external application as well. It also supports versioning, document metadata. Advanced Sharepoint-like actions like commit message during checkin is not yet supported.

It has pluggable workflow integration, the following engines are supported out of the box:

- jBPM3
- Kaleo

The latter one is configured by default, and it even has a few sample definitions after installation. Unfortunately, it seems Liferay invented its own schema for process definitions when using Kaleo [43], and thus does not provide any support for the standard BPMN format.

Its office integration is solely due to the WebDAV interface, with its known limitations for our purpose.

Feature	**Sharepoint Workflow**	**Liferay**	**jBPM**
License	proprietary	open-source	open-source
Decoupling	no	no	yes
Office integration	yes	no	no
Standard process definition format	no	no	yes

Table 7.2. Comparison of related workflow solutions

[2]The exact version I evaluated: liferay-portal-tomcat-6.0.6-20110225.

Chapter 8

Conclusion and Future Work

The solution designed shows a way to integrate open-source office productivity software in an enterprise environment. We described how today's document management servers and workflow management systems work, designed and developed office integration to document management systems, developed office integration of workflow support, finally we evaluated the solution on multiple platforms.

8.1 Document management

We finished the design of the solution, and it was clear that creating a LibreOffice extension that provides the necessary features is certainly possible. We also implemented support for the most important use-cases:

- workspace, folder and document management based on the folder/file metaphor
- version control support for documents: creating, updating, listing and deleting versions
- collaboration during document editing: lock management

As already discussed above, the most important weakness of the solution is that it focuses only on the core functionality:

- As part of error handling, it presents permission errors to the user, but changing permissions is not possible from the extension.
- Managing users, tasks and links is yet to be designed and implemented.

Not ignoring the shortcomings, we can still conclude that the created solution makes migrating office productivity suites to open-source alternatives easier, which was our initial goal.

There are some implementation and design issues that need to be addressed, however, in the future:

- An implementation improvement would be to review the inherited codebase and make sure that the user interface runs in a separate thread in all cases. Currently, there are cases when the user interface is not repainted due to waiting for an input from the user.
- Using LibreOffice's filepickers instead of reinventing our own would result in a more consistent user experience.
- Once CMIS will be widely implemented by proprietary document management servers, use the CMIS protocol instead of the native SharePoint for client-server communication.

8.2 Workflows

Once the design of the document management part was ready, we continued planning the workflow integration part. Our approach shows, how a workflow engine and an existing document-management server can be successfully integrated into open-source office suites, addressing several use-cases originating from an enterprise environment:

- Management of personal and group-assigned human tasks.
- Decoupled access of the document server and the workflow engine.
- Masked document access, based on task information.
- Support of business decisions, affecting the associated process instance.
- Audit log integration.

The following features are not designed in the current thesis, but they could be done in the future:

- The gwt-console-server already provides a REST API to access the audit log, the gwt-console native UI could be improved to take advantage of that.
- The current extension can communicate with jBPM only, it could be extended to support various other pluggable workflow engines.

The following features are partly provided by our extension, but could be improved further:

- Extending business decision support, adding integration for other kinds of decisions (see Figure 3.6 and Figure 2.10) would be helpful.

- A document can be attached to a single process instance only, this could be extended to allow multiple process instances accessing the same document in parallel – but in that case proper locking should be designed first.

To summarize our results:

- We designed client-side document management integration.
- We added support for workflow integration.
- We kept the created solution portable across different platforms.

We hope that the release of our LibreOffice extension as an open-source component contributes in general to the acceptance of open-source software in enterprise environments.

Bibliography

[1] Subversion: Official website. http://subversion.apache.org/.

[2] Git: Fast Version Control System. http://git-scm.com/.

[3] Microsoft Office. http://office.microsoft.com/en-us/.

[4] The Document Foundation: LibreOffice. http://www.libreoffice.org/.

[5] OpenOffice.org - The Free and Open Productivity Suite. http://www.openoffice.org/.

[6] Microsoft: SharePoint Official Site. http://sharepoint.microsoft.com/en-us/Pages/default.aspx.

[7] Alfresco: Open Source Enterprise Content Management System. http://www.alfresco.com/.

[8] Google Docs. http://docs.google.com/.

[9] The CMIS v1.0 OASIS Standard Specification. http://docs.oasis-open.org/cmis/CMIS/v1.0/os/cmis-spec-v1.0.pdf.

[10] CMSWire: IBM's Social Strategy Includes Support for CMIS. http://tinyurl.com/ibm-cmis.

[11] Alfresco Wiki: Main Page. http://wiki.alfresco.com/wiki/Main_Page.

[12] Microsoft Sharepoint: put document Method, document Parameter. http://msdn.microsoft.com/en-us/library/ms416274.aspx.

[13] Joining Dots: Blog: SharePoint History. http://www.joiningdots.net/blog/2006/08/sharepoint-history.html.

[14] OpenOffice.org Wiki: Uno. http://tinyurl.com/wiki-uno.

[15] Apache: The HTTP Server Project. http://httpd.apache.org/.

[16] Oracle: javax.servlet Package (Java EE 6. http://download.oracle.com/javaee/6/api/javax/servlet/package-summary.html.

[17] Jason Greene: Why is JBoss AS 7 so fast? http://planet.jboss.org/post/why_is_jboss_as_7_so_fast.

[18] Object Management Group: Business Process Management Initiative. http://www.bpmn.org/.

[19] JBoss Community: jBPM. http://www.jboss.org/jbpm.

[20] Eclipsepedia: What databases are supported by JPA. http://wiki.eclipse.org/EclipseLink/FAQ/JPA.

[21] John O'Conner: Creating Extensible Applications With the Java Platform (Sun Developer Network). http://java.sun.com/developer/technicalArticles/javase/extensible/.

[22] AlfrescoForge: OpenOffice.org Plugin for Alfresco. http://forge.alfresco.com/projects/opal/.

[23] Microsoft Sharepoint: Method Syntax. http://msdn.microsoft.com/en-us/library/ms463030.aspx.

[24] Wireshark: Go deep. http://www.wireshark.org/.

[25] Apache HttpClient Home. http://hc.apache.org/httpclient-legacy/.

[26] Apache HttpComponents. http://hc.apache.org/.

[27] TagSoup. http://home.ccil.org/~cowan/XML/tagsoup/.

[28] XForms 1.1: Serialization as application/x-www-form-urlencoded. http://www.w3.org/TR/xforms/#serialize-urlencode.

[29] Douglas Crockford: The application/json Media Type for JavaScript Object Notation (JSON). http://tools.ietf.org/html/rfc4627.

[30] Google Project Hosting: A Java library to convert JSON to Java objects and vice-versa. http://code.google.com/p/google-gson/.

[31] JSON Format Home Page: Java. http://json.org/java/.

[32] LibreOffice Reference: Framework Module. http://opengrok.libreoffice.org/xref/core/framework/inc/protocols.h#50.

[33] LibreOffice CMIS. http://gitorious.org/libreoffice-cmis.

[34] Oracle: Connector for SharePoint Server. http://extensions.services.openoffice.org/en/project/sharepoint_connector.

[35] Cédric Bosdonnat: LibCMIS. https://gitorious.org/libcmis.

[36] Drupal: SharePoint Project. http://drupal.org/project/sharepoint.

[37] J Wang: A framework for document-driven workflow systems. http://php.scripts.psu.edu/faculty/a/x/axk41/BPM05-jerry-reprint.pdf.

[38] R Carbon, G Johann, T Keuler, D Muthig: Mobility in the virtual office: a document-centric workflow approach. https://mailserver.di.unipi.it/ricerca/proceedings/ICSE2008/sam/p21.pdf.

[39] R Krishnan, L Munaga: XDoC-WFMS, A Framework for Document Centric Workflow Management System. http://www.cs.sunysb.edu/~krishnan/xdoc.htm.

[40] Access control in document-centric workflow systems – an agent-based approach. http://ce.sharif.edu/~yuosefsa/article/CS.pdf.

[41] Microsoft: SharePoint Designer. http://www.microsoft.com/download/en/details.aspx?id=21581.

[42] Liferay: Enterprise open source portal and collaboration software. http://www.liferay.com/.

[43] Liferay Portal Documentation: Workflow with Kaleo. http://www.liferay.com/documentation/liferay-portal/6.0/administration.