



**Hochschule
Bonn-Rhein-Sieg**
University of Applied Sciences

Fachbereich Informatik
Computer Science Department

Bachelor Thesis

Bachelor Degree Course Business Information Systems

An Approach to the Integration and Use of YAWL in a Business Portal Environment

Provided by Christian Freihoff

Examination Committee: Prof. Dr. Andreas Hense (Supervisor)
Prof. Dr. Rüdiger Buck-Emden

Date of Submission: July 10, 2014

Declaration

I hereby declare, that the work presented in this thesis is solely my work and that to the best of my knowledge this work is original, except where indicated by references to other authors.

This thesis has neither been submitted to another committee, nor has it been published before.

Sankt Augustin, July 10, 2014

Christian Freihoff

Acknowledgments

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Dr. Andreas Hense for the continuous support of my study and research and to Prof. Dr. Rüdiger Buck-Emden for being available as second examiner.

My sincere thanks also goes to the current and former members of the Rheni GmbH team from whom I have learned a lot about the development of Java and web applications.

I also would like to thank Arely Sanchez for the quick and competent proofreading of this thesis.

Last but not least, I want to thank my beloved wife Melanie for encouraging me to start studying and for providing so much power, support and patience during the last years.

I want to dedicate this thesis to my father who was not allowed to see the completion of my study.

Contents

1. Introduction	1
1.1. Business Process Automation	1
1.1.1. Overview and General Aspects	1
1.1.2. Technical Aspects	2
1.2. YAWL - Yet Another Workflow Language	5
1.3. Business Portal Environment	7
1.4. Subject of Examination	7
1.5. State of The Art	8
1.6. Outline of Examination	9
2. The Examination Environment Components	10
2.1. Application Server Environment	10
2.2. Custom Data Type Definition for YAWL	10
2.3. External Organizational Data	11
2.3.1. The Different Organizational Models	13
2.3.2. Organizational Data Mapping	13
2.4. Codelets as YAWL Workflow Interfaces	13
2.5. Persistence via the Hyperjaxb3 Project	16
2.6. Liferay Portal	17
2.7. Portlet Applications with Vaadin	19
2.7.1. Anatomy of a Portlet Application	19
2.7.2. The Vaadin Framework	20
2.8. Overall Architecture	21
2.8.1. Excursus: Technical Architecture Modeling	22
2.8.2. Expected overall architecture	23
3. Integration Approach	25
3.1. Installation of Liferay Portal and YAWL on one Application Server	26
3.1.1. Liferay Portal	26
3.1.2. YAWL WMS	29
3.1.3. Additional Dependencies and Conclusion	30
3.2. Definition of Sample Workflow and Data Type	31
3.3. Implementation of the Persistence Layer	35
3.3.1. Generation of Entity Classes	35
3.3.2. Database Configuration and Data Access Objects	37
3.3.3. Conclusion and Deployment	42
3.4. YAWL Extensions	42
3.4.1. Organizational Data Import	43
3.4.2. Liferay Portal Orgdata Class	43
3.4.3. Liferay Resources Adapter Class	45
3.4.4. Persistence Codelets	47
3.4.5. Portal Communication Codelet	49

Contents

3.4.6. Style Adjustment by CSS	52
3.4.7. Conclusion and Deployment	52
3.5. Implementation of Portlet Applications	55
3.5.1. Generic Portlet Body	56
3.5.2. Workqueue Portlet	58
3.5.3. Workitem Portlet	59
3.5.4. Data Overview Portlet	59
3.5.5. Data Details Portlet	63
3.5.6. Auxiliary Classes	63
3.5.7. Deployment	67
3.6. Acute Issues During the Integration	70
3.6.1. Liferay Hibernate Cache	70
3.6.2. YAWL Issues	71
3.6.3. Conclusion	72
4. Results and Evaluation	72
4.1. Used Technologies	73
4.2. Interoperability of YAWL	74
4.3. Productive Usability of YAWL	75
4.4. Limitations	75
4.5. Conclusion	75
4.6. Recommendations	76
Bibliography	77
A. Contents of the Attached Data Carrier	83
A.1. Root Folder	83
A.2. Literature	83
A.2.1. Online Backups	83
A.3. Illustrations	83
A.4. Software	83
A.4.1. LiferayAddons	83
A.4.2. PersistenceLayer	83
A.4.3. Portlets	83
A.4.4. Prepared Bundle	83
A.4.5. ResourceServiceLibs	84
A.4.6. Specifications	84
A.4.7. YawlCoreServices	84
A.4.8. YawlExtensions	84
B. Installation Manuals	85
B.1. Download URLs	85
B.2. Expert Setup	85
B.3. Prepared Bundle Setup	87
B.4. Development Setup	88

1. Introduction

Today, we face a world with a globalized economy. Almost every market on this planet is now reachable for those companies who were only national or regional providers some decades ago. This means promising prospects for such a company, but also for all these companies on the globe. The number of products which can be produced almost all over the world in every desired quality or design is constantly increasing. Thereby, previously important factors like quality have lost importance as unique selling points, and companies are forced into new competitive situations where the advantages are given by efficiency, cost reduction, and technology. This is exactly the area of application of advanced automated business processes which consequently represent crucial success factors for companies who compete globally.

Another aspect is the increasing use of business portals, which have gained considerable importance within the last years. Accumulations of different programs which have been running on local devices, or at best within local networks, are increasingly replaced by browser-based applications which can be globally accessed from any place with an Internet connection. Because business portals are usually operated on external virtual servers, they have a high potential to reduce costs. Capacities are highly flexible and only the currently required capacity must be paid, while on self-hosted physical servers capacity bottlenecks, and expensive idle times alternate with each other. The systematic and consistent way applications are integrated in such a portal can significantly increase efficiency.

Both of the aforementioned technologies already present a potential competitive advantage for themselves. An integration or interlinking of these technologies should therefore lead to a correspondingly higher potential for efficiency and cost reduction.

1.1. Business Process Automation

1.1.1. Overview and General Aspects

In [RW12, p. 3] Reichert and Weber define a business process “as a set of one or more connected activities which collectively realize a particular business goal”. The Terminology & Glossary of the Workflow Management Coalition defines:

“A business process is typically associated with operational objectives and business relationships, for example an Insurance Claims Process, or Engineering Development Process. A process may be wholly contained within a single organizational unit or may span several different organizations, such as in a customer-supplier relationship.”

Florian Gottschalk, who developed the configurable workflow language C-YAWL, defines business process management (BPM) as “the field that is concerned with the orchestration of individual tasks to executable processes” and writes further [Got09, p. 1]:

1. Introduction

“By documenting current or future options available to handle data, by defining the execution order of tasks, as well as by depicting possible outcomes, process models help stakeholders when developing, implementing, executing, or improving business processes. [...] Moreover, process models can also serve as instructions for information systems that support the data processing during the execution of business processes, known as workflow management systems”.

Reichert and Weber also claim that any automation of business processes should be preceded by designing and optimizing business processes to meet organizational goals in an economic and effective manner [RW12, cf. p. 9]. This means that the management of business processes does not necessarily appear in union with business process automation, but the management of business processes is an indispensable predecessor for their automation. A business process model, as an outcome of Business Process Management, describes a business process at a high level of abstraction and includes all aspects which are necessary to visualize, simulate, and analyze the process from a business perspective (e.g., cost and time). Such a process model may include individual activities which are manual and can not be automated. Hence, an executable counterpart is needed for the automation of a business process. One name for this counterpart is the workflow model, which represents the automation of a business process and the implementation of its model [RW12, cf. p. 9f.].

Van der Aalst et al. use technological aspects to distinguish management from automation of business processes. They describe the application of business process management rather as “pen-and-paper” exercises, such as the visualization and improvement of process models, while Business Process Automation means the use of precise process descriptions for guiding the performance of business activities. When it needs to be executed, work is delivered to selected resources. These can be either humans or software applications [HvdAAR10, cf. p. 3].

A workflow is defined as “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules” [Coa99, p. 8]. Consequently, a workflow can be seen as the technical or IT-based implementation of an automated business process and is based on an executable process model.

1.1.2. Technical Aspects

From Business Process to Workflow

As discussed above, a business process requires an executable model or definition to be processed as a workflow. An important additional component is the IT system on which the workflow is executed. Such a system is called a workflow management system (WM-S/WfMS) [Coa99, HvdAAR10] or process aware information system (PAIS) [RW12]. In the following, the term workflow management system (WMS) will be used exclusively. The Workflow Management Coalition defines a WMS as

“a system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications” [Coa99, p. 9].

1. Introduction

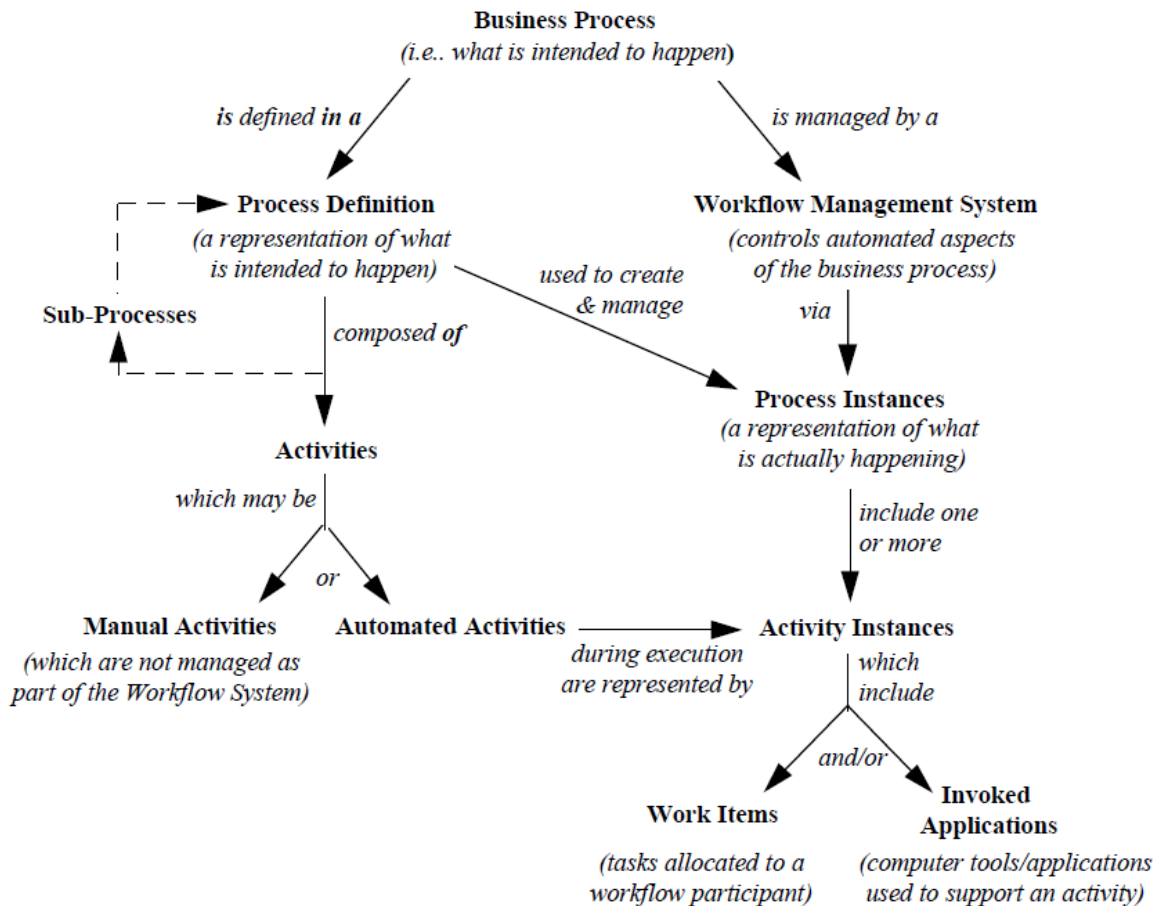


Figure 1.1.: Relationships between process and workflow [Coa99, p. 7]

Figure 1.1 shows the relationship between business process and workflow. The business process pretends what is intended to happen from a business point of view. The process definition is defined as

“the representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc” [Coa99, p. 11]

So, it represents the executable definition of this process and contains all technical aspects and details needed for automation. The Workflow Management System uses the definition to create instances of the automated process (i.e. workflows). The process definition is composed of one or more activities (which can be automated or manual). Inside a running workflow instance, these activities are represented by instances of themselves. Figure 1.2 clarifies the relationships between the defined process components at build time and their instances at runtime.

1. Introduction

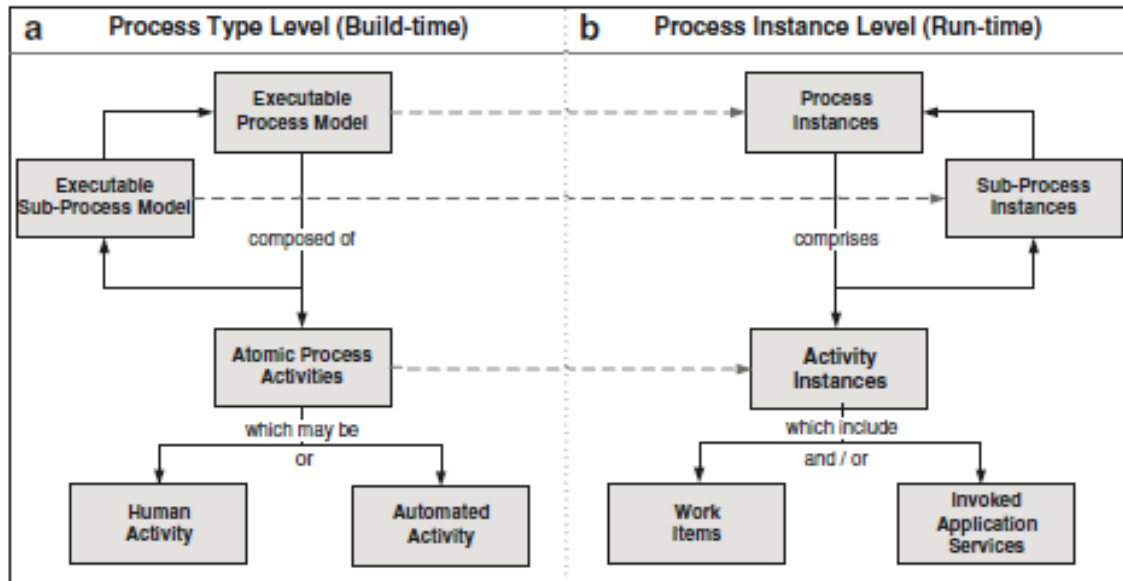


Figure 1.2.: Relationships between build time and runtime [RW12, p. 31]

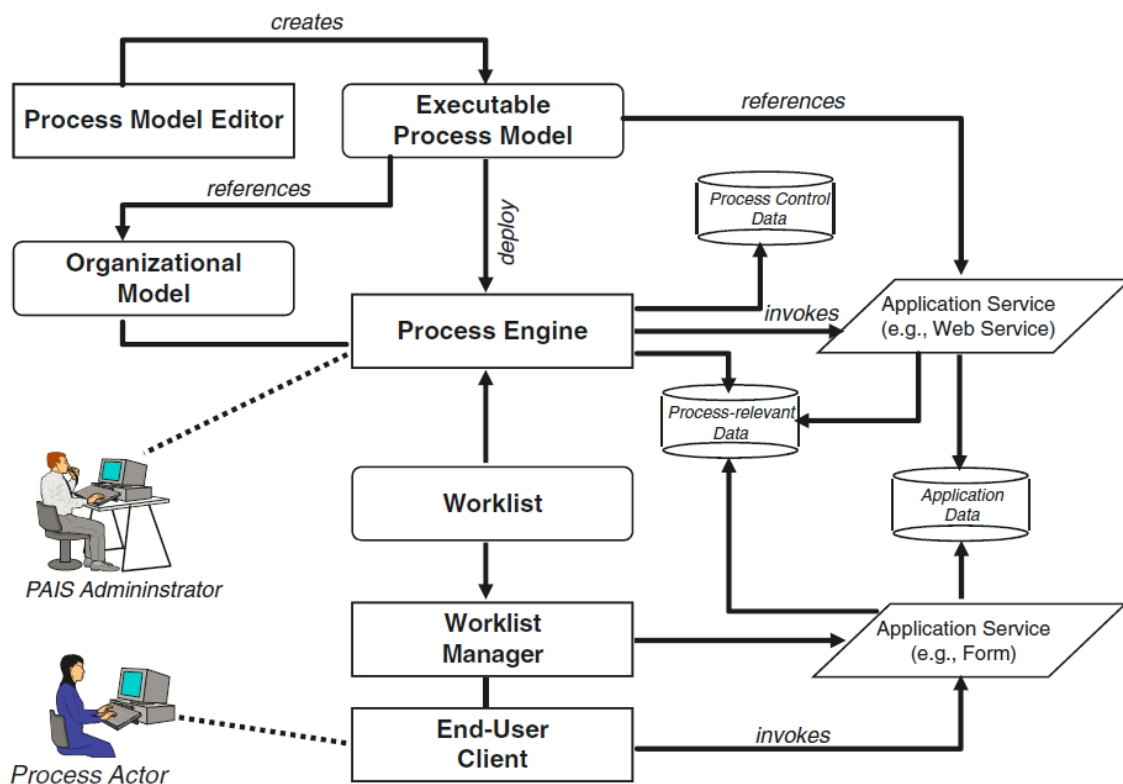


Figure 1.3.: Basic components of a WMS denoted as PAIS in [RW12, p. 32]

1. Introduction

Components of a Workflow Management System

A regular WMS consists of several basic components. The only one which is not mandatory at runtime is the process model editor. This component is used to create the above-mentioned executable process model (i.e. the process definition) and may include tools for graphic design or validation.

After creation, the process model is deployed to the core component of a WMS called Process Engine [RW12] or Workflow Engine [Coa99, HvdAAR10]. This engine governs process-relevant data (e.g. the current state of a single activity), invokes any required services, and manages the worklist.

The Worklist contains instances of activities (in the following denoted as work items) which are associated with a given user who participates in the workflow [Coa99, cf. p. 20]. In the following, such users are denoted as participants, who interact with the End-User Client, to work on their assigned work items. Worklist and End-User Client are connected by a component denoted as Worklist Manager [RW12] or Worklist Handler [Coa99, HvdAAR10]. It is defined as a

“software component that manages the interaction between the user (or group of users) and the worklist maintained by a workflow engine. It enables work items to be passed from the workflow management system to users and notifications of completion or other work status conditions to be passed between the user and the workflow management system” [Coa99, p. 21].

Figure 1.3 gives an overview of the components described above. As can be seen, there is no component of the WMS which can access the application data. This fact will play a role in the further course of this examination.

1.2. YAWL - Yet Another Workflow Language

YAWL, an acronym for *Yet Another Workflow Language*, is “a workflow modeling language inspired by Petri Nets but with several important extensions and its own semantics” [Got09, p. 36]. It represents an outcome of a joint effort between the Eindhoven University of Technology and the Queensland University of Technology. The development of YAWL was inspired by the evaluation of several contemporary existing workflow products. All of those had different abilities and limitations and none of them supported all 20 relevant workflow control-flow patterns which had been identified in previous researches [vdAtH05].

“The language YAWL was designed, based on Petri nets as to preserve their strengths for the specification of control-flow dependencies in workflows, with extensions that allowed for straightforward specification of these patterns. YAWL can be considered a very powerful workflow language, built upon experiences with languages supported by contemporary workflow management systems. [...] Its design hopefully allows YAWL to be used for the purposes of the study of expressiveness and interoperability issues.”[vdAtH05, p. 38]

After several years of further development, YAWL is much more than just a reference implementation which supports the most relevant workflow control-flow patterns. YAWL has grown into a full-fledged workflow management system and is used in a wide variety of academic and industrial settings. The term YAWL is synonymous with both the language and

1. Introduction

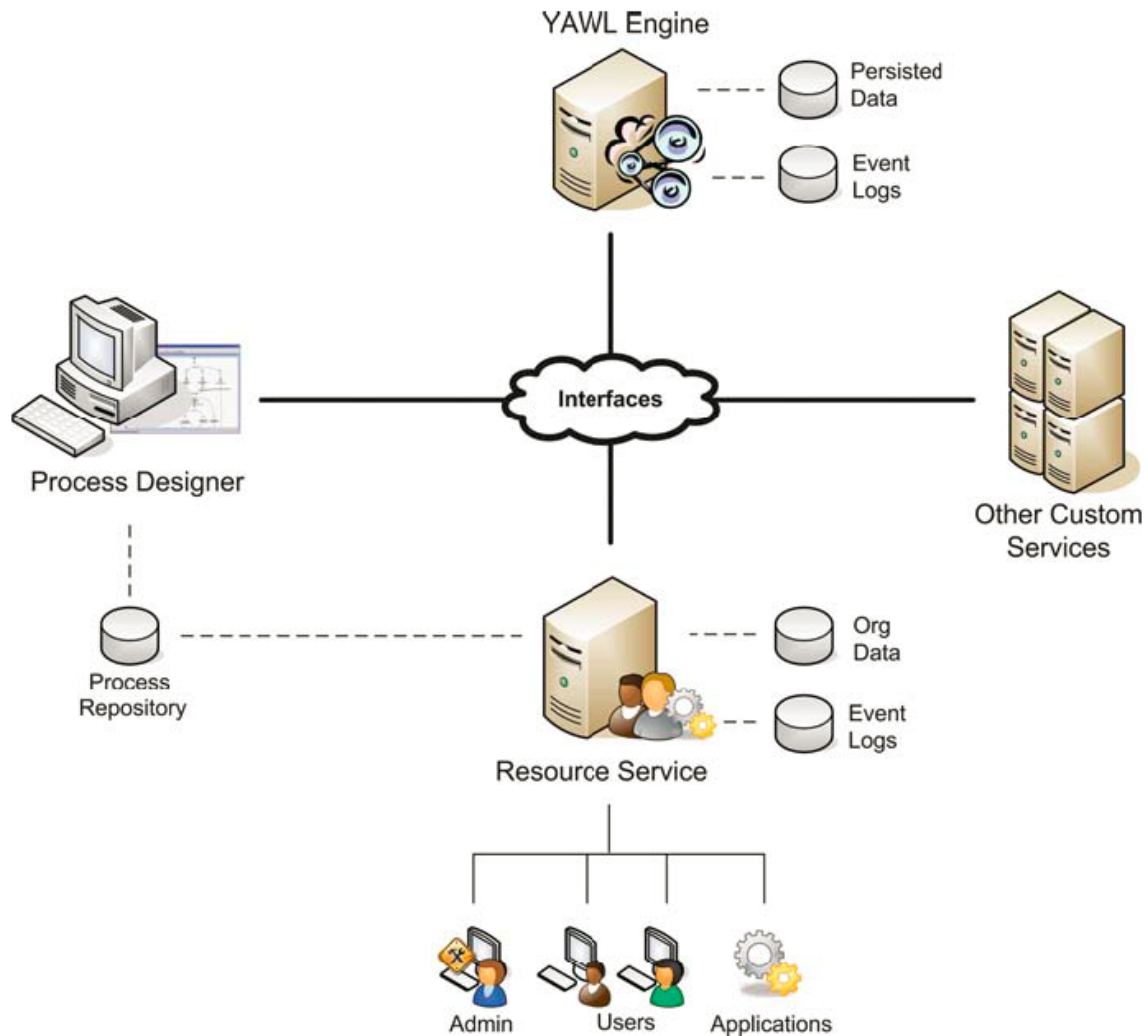


Figure 1.4.: Simplified architectural overview of the YAWL environment[HvdAAR10, p. 15]

the support environment[HvdAAR10, cf. p. 11]. Because YAWL is an outcome of academic research, it could be developed without any commercial interests and is a completely open source. Hence, it is not only free of charge but can also be extended or changed according to individual requirements. During the development part of this examination, this option has been applied. The high expressiveness and a high level of maturity, combined with the open source aspect, position YAWL uniquely in the crowded field of workflow languages and workflow management systems.

In figure 1.4 a simplified overview of the YAWL environment is provided. The process designer, in the following denoted as YAWL editor, is used to create the workflow specifications (i.e. the executable process model). The current versions 2.3.5 and 3.0beta provide a convenient graphical user interface for rapid prototyping and process validation. The YAWL Engine represents the core of the WMS. It is responsible for executing process instances, preparing work items (i.e., task instances) and for appropriate timing to comply with the specified control-flow. The engine is not responsible for how or by whom a work item is

1. Introduction

processed. It only delivers work items to a specified external service at the specified time (or state of control-flow) and takes them back after processing [HvdAAR10, cf. p. 241f.]. Such a service can be one of the included ones or a self-implemented extension. In this examination, only the Resource Service is relevant. Therefore, the other custom services are not discussed further.

The Resource Service is aware of the respective workflow specification and the organizational data (i.e., users, roles and related data). It manages the worklist and provides access for users, applications, and administration. The YAWL Resource Service also provides capabilities to process work items. These are generated forms for manual processing and interfaces for automated handling (denoted as codelets). So, the Resource Service is the central point of contact in the YAWL WMS and therefore an eminently important component.

1.3. Business Portal Environment

Sezov Jr. defines a Portal as a “web-based gateway that allows users to locate and create new relevant content and use applications they commonly need to be productive” [SK12, p. 4] and further as a

“single web-based environment from which all of a user’s applications can run. These applications are integrated together in a consistent and systematic way” [SK12, p. 5].

“A portal is a collection of mini web applications, called portlets. A portal supports features like personalization, content aggregation, authentication, and customization. Portlets act as windowed web applications within the portal, and each window in a portal web page (called a portlet page) represents a portlet” [Sar12, p. 4].

So, a portal environment is a web application containing several other web applications, which are denoted as portlets. The portal provides a uniform appearance for all contained portlets, a central management for authentication, and organizational data (e.g. users, roles). That means a portal user has access to all data which is relevant for his work and to all applications he is authorized to use. If the portal has been consistently and fully implemented, the user can do all his work within the portal environment. A portal that serves commercial purposes is denoted as business portal.

1.4. Subject of Examination

Since it has been introduced in 2005 [vdAtH05], YAWL has been successively developed further. New capabilities have been added and old limitations and bugs have been removed. Nevertheless, with every new extension or improvement new bugs and limitations find their way into this WMS. The mostly inadequate documentation and the small community of users lead to a poor support. So, YAWL still continues to be a system in ongoing development. On the other hand, YAWL is the only open source workflow management system which provides such a range of flexibility, expressiveness, and functionality. Hence, YAWL should be the first choice of companies and organizations looking for a non-commercial application

1. Introduction

to automate their business processes.

When examining the integration of an open source WMS into a business portal environment, it is obvious that this portal should also be an open source solution which is free of charge and can be extended or changed if necessary. Because of expected limitations and issues in the YAWL WMS, the used portal should provide a high level of maturity and a comprehensive support. Liferay Portal meets these requirements and is therefore selected as the business portal environment into which the YAWL WMS shall be integrated.

This examination shall answer the following questions: Can YAWL be integrated to Liferay Portal and applied there as a workflow management system, i.e., can users of the portal access and process their assigned work items out of a running workflow inside the portal environment? Can workflows deployed to the YAWL WMS be triggered by a portal application? Can a YAWL workflow and a portal application share the same data on the same database without causing inconsistency?

The focus of this examination is on the interoperability of the YAWL WMS. Liferay Portal merely represents a sample business portal environment for the integration of YAWL. Therefore, Liferay specific details will only be considered if necessary.

1.5. State of The Art

The workflow management system YAWL and its interoperability with a portal has already been the subject of three current master theses. In 2013 M. Roskosch developed a management system for IT configuration items. These can be checked in, sent to another location, received from another location, etc. which is all done via YAWL workflows. Additionally the data of the configuration items can be viewed and edited via special portlet applications running in Liferay portal [Ros13].

An approach with less application orientation but with a closer look on development and automation aspects have been recently examined in two further masters theses. M. Dames and D. Morosjuk independently investigated how YAWL and Liferay Portal applications can access the same database with a minimum of manual development. They developed two different code generators which create persistence layers for YAWL and Liferay as well as rudimentary portlet applications and resource sharing between the two systems [Mor14, Dam14]. In both approaches, some limitations or disadvantages were left. These may prevent a derivation to a generic application model capable for productive use.

Apart from the fact that M. Roskosch reported problems with the Liferay service builder (which automatically creates a persistence layer for Liferay content) [Ros13, cf. p. 61], in his application, the connection between YAWL and the database for collected data is almost completely achieved by manually programmed interfaces. So, when the data model changes (what obviously has to be expected in a different application context), all these interfaces have to be changed or rewritten from scratch. Moreover, the capabilities of the Liferay service builder are evaluated as not sufficient compared to the data type capabilities of YAWL [Ros13, cf. p. 63ff.].

1. Introduction

Dames and Morosjuk avoided this lack of flexibility in manually implemented interfaces for YAWL by generating the whole persistence layer for both the portal and the WMS automatically [Dam14, Mor14]. Because this is done based on the respective data structure the application context pretends, this approach is much more flexible and less error prone. However, with that approach new issues appear. Liferay has a progressive caching mechanism for the portal content, and both Dames and Morosjuk persist their workflow-collected data as portal content to Liferay's database. Because the data is written to the portal's database without notifying the content management, Liferay does not reload the cached data until it expires after up to ten minutes. In the worst case, the portal has to be restarted to load the new-added data [Dam14, cf. p. 79] which is obviously a massive limitation for a productive operation. Morosjuk evaluates the use of automatically created portlets as not sufficient because they only provide a simple CRUD functionality (i.e., a rudimentary database access and manipulation mechanism) [Mor14, cf. p. 66].

1.6. Outline of Examination

In the main, this bachelor thesis takes a different approach than the previous master theses. The goal is an application sample that allows a derivation to productive usable models in different application contexts. For that it is attempted to circumvent or eliminate all known issues or limitations.

Based on a sample business process and a simple organizational model (including respective users and roles), a YAWL workflow is created and configured to use the business objects (i.e., the data structure) pretended by the sample process.

A sample business portal is set up and filled with the above-mentioned users and roles. Because the portal shall manage the organizational data exclusively, the WMS must be able to access the respective data in the portal environment. While in previous examinations YAWL just read out the portal's database, in this thesis a more sophisticated way is gone. Both applications, YAWL and Liferay, do not touch each other's databases. Instead they use the provided service classes to communicate with each other what results in a clean and proper integration of YAWL in the business portal environment.

The business data (i.e., the instances of the defined business objects) are persisted in a separate enterprise database and not as portal content or in one of the application's databases. This database will be accessed by a uniform persistence layer which can be used by YAWL workflows and portal applications equally. To provide adequate genericity and lower error rate, as many components of this persistence layer as possible shall be generated automatically.

Because one of the outcomes of this examination is a realistic business application sample, the portlet development focuses on functionality, user friendliness, and convenience. The portlet applications shall provide functions and optical appearance which correspond with the business context and the expectations of the users as they might be.

At first, the required components for the integration approach are identified and selected. This is done respectively to the requirements mentioned above. The subsequent integra-

tion approach describes and documents the implementation and integration of the identified selected components. All important and specific implementation components are listed and discussed in detail. Of course, upcoming issues and problems are documented as well. Thereafter, the evaluation discusses the used technologies as well as the interoperability of YAWL and the portal application. Recommendations and open issues provide prospects for future research.

2. The Examination Environment Components

2.1. Application Server Environment

Both the YAWL WMS and Liferay Portal are web applications which have to be deployed to a web application server like Apache Tomcat. The interface classes provided by YAWL can be used from any place with a network connection to a running YAWL instance. In contrast, the service APIs of Liferay can only be accessed from the same web application server. Having two server instances running for two applications which are interacting with each other means redundancy. If these two servers run on different physical machines or virtual machines, there are also security issues which have to be considered. The YAWL WMS is satisfied with little resources on its server while Liferay Portal in its current version requires a significant amount of resources. So the Liferay Portal web server should have no problem with carrying an additional (and comparatively lightweight) application like YAWL. Finally, there are many reasons for running the WMS and the portal on the same server, and there are no rational reasons against this. YAWL and Liferay are available bundled with Apache Tomcat. So exactly this server is selected to carry both applications, the version is decided by the highest one of both Tomcat servers.

2.2. Custom Data Type Definition for YAWL

The YAWL workflow modeling language offers all the usual primitive data types (e.g. String, boolean) from scratch. This may be sufficient for trying out simple workflows, but not for business application contexts where the objects can have an enormous complexity, including subtypes and relationships. As a progressive modeling language, YAWL offers the possibility to use self-defined complex data types using XML schema definition (XSD) [HvdAAR10, cf. pp. 93ff.].

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="person">
    <xs:sequence>
      <xs:element name="firstName" type="String"/>
      <xs:element name="lastName" type="String"/>
      <xs:element name="adress" type="adressType"/>
    
```

2. The Examination Environment Components

```
</xs:sequence>
</xs:complexType>

<xs:complexType name="adressType">
  <xs:sequence>
    <xs:element name="street" type="String"/>
    <xs:element name="number" type="String"/>
    <xs:element name="city" type="String"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Listing 2.1: Sample XML schema definition

The listing 2.1 provides a small sample of a complex data type defined via XML schema. As can be seen, the type "person" contains some primitive String elements and again another complex type. Such a nesting can be applied as often as needed and at any depth level. This way, YAWL allows the use of almost every data structure or business object including, one-to-many mappings and restrictions (e.g. a pretended range for a date value). The aspired application model shall be derivable to as many different business contexts as possible, so the data type definition mechanism needs to be capable for structures with a high level of complexity. In addition, YAWL itself and the later discussed Hyperjaxb3 project make extensive use of XSD-based definitions and variables. So, the use of XML schema defined data types is the way to go for this examination.

2.3. External Organizational Data

By default, both applications, the YAWL WMS and Liferay Portal, have an organizational data management. When these two systems shall interact with each other, this data (e.g. roles, users) has to match in both applications exactly. This would result in either two redundant data managements, accompanied by expenses twice as high and a much higher error rate, or it would mean to use only one organizational data management and to provide this data to the other system.

Because it is intended to integrate YAWL into Liferay and not the other way round, it is obvious that the organizational data (in the following also denoted as orgdata) has to be managed exclusively by the portal. Fortunately, YAWL offers a mechanism to import external orgdata by reimplementing or replacing the respective Java class. Listing 2.2 shows an excerpt of the web.xml file which belongs to YAWL's Resource Service.

```
...

<context-param>
  <param-name>OrgDataSource</param-name>
  <param-value>HibernateImpl</param-value>
  <description>The name of the class (an extension of 'DataSource') which is
    used to as the 'data translation layer' between the ResourceService
```

2. The Examination Environment Components

```
        and a source of organisational data. The default YAWL implementation is
        the 'HibernateImpl' class.
    </description>
</context-param>

<context-param>
    <param-name>ExternalUserAuthentication</param-name>
    <param-value>>false</param-value>
    <description>When set to false (the default), user passwords are encrypted
        and stored within each user record, and authentication is handled
        within the resource service. When set to true, user authentication is
        deferred to the currently implemented external org data source, and
        passwords are sent to the data source as plain text (rather than the
        default encryption). It is up to the external data source provide valid
        user authentication in this case. A setting of true is only relevant
        if an external data source is active; if the default YAWL org database
        is in use, this setting is ignored.
    </description>
</context-param>

...

<context-param>
    <param-name>OrgDataRefreshRate</param-name>
    <param-value>-1</param-value>
    <description>The number of minutes delay between refreshes of org data
        from data store. That is, every X minutes reload the org data from the
        data store into the ResourceService. If the org data source is updated
        externally, specify how often the data should be refreshed. A setting
        of -1 disables the refresh (this is the default, since the YAWL org
        data source is not modified externally). Common settings: 1440 = 24
        hours; 360 = 6 hours; 10080 = 1 week.
    </description>
</context-param>

...
```

Listing 2.2: Resource Service web.xml

By default, the Resource Service obtains the organizational data from its own database via the *HibernateImpl* class as defined in the first parameter in listing 2.2. The developer can change this setting to another Java class, which extends YAWL's *DataSource* class and provides the respective methods. So, it is possible to implement a Java class which accesses the portal's orgdata management to obtain users and roles. In addition, an external user authentication can be implemented and activated (second context setting). Of course, in productive use the organizational data inside the portal will change (e.g. because of new added users or new assigned roles), and these changes have to be propagated to the WMS. For that reason, the YAWL developers added a polling mechanism which can be activated by setting the *OrgDataRefreshRate* (third context setting) to a minute-valued interval. By use

2. The Examination Environment Components

of this mechanism, the whole organizational data can be stored in the portal and managed there exclusively. YAWL can still access this data to allow user logins, or to assign work items to the respective user.

2.3.1. The Different Organizational Models

As might be expected, the organizational models of YAWL and Liferay Portal only match partially. Regarding the individual users, there is no difference at first appearance apart from the term. “In the Resource Service, a human resource is referred to as a participant, that is, someone who willingly participates in the performance of tasks within a workflow instance, progressing it towards completion” [HvdAAR10, p. 265]. So, when fetching the organizational data from Liferay Portal and assigning its components to their corresponding counterpart, a portal user can be mapped to a YAWL participant one to one.

As can be seen in tables 2.1 and 2.3, a role is also very similar to its equivalent, but two differences can be figured out. In YAWL, a role can belong to another superordinate role which is not possible in the portal. On the other hand, the Liferay roles can be classified in categories which are not provided by the YAWL WMS. The situation is similar with Liferay’s orgdata component called organization. Its counterpart, the YAWL *Org Group* can also be used to represent a hierarchical structure of the organization. Community and a user group in Liferay Portal on one hand, and YAWL’s capability and position on the other hand do not find a useful match. The user privileges in the YAWL organizational model are a special case. Because they represent permissions how to handle a work item at runtime, it is strongly recommended to set them when the orgdata is fetched from the portal.

2.3.2. Organizational Data Mapping

Figure 2.1 shows the organizational data mappings as determined for this examination. As previously stated, a Liferay Portal user is mapped to a YAWL participant. A role in the portal is represented by a role in the WMS. The categorization of roles in Liferay and the availability of superordinate roles in YAWL are not taken into account. Because there is absolutely no equivalent for a user’s privileges in Liferay, they are set depending on the role. That means that if a portal user has an administrative role, his participant counterpart obtains all available privileges. If a participant is mapped from a user with a non-administrative role, only some privileges are granted. Because they are not absolutely necessary, all other components of the organizational models are left out in this examination.

2.4. Codelets as YAWL Workflow Interfaces

One possibility of YAWL to access workflow-handled data in external databases are YAWL’s Data Gateways, which allow to populate task input variables and to update fields in the database from task output variables. In his master thesis [Dam14] M. Dames was able to determine that these Data Gateways are not sufficiently reliable [Dam14, cf. p. 25]. So, for an integration approach which shall represent a model for productive usable business applications, this way of database access is absolutely out of the question. YAWL offers another

2. The Examination Environment Components

Table 2.1.: Liferay's Organizational Model [SK12, p. 19]

Type	Description
Role	Collects users by their function. Permissions in the portal can be attached to roles.
Organization	Collects users by their position in a hierarchy. Organizations can be nested in a tree structure. You use organizations to represent things like a company's organizational chart.
Community	Collects users who have a common interest. Communities are single entities and can't be grouped hierarchically. By default, users can join and leave communities whenever they want, although you can administratively change this so users are assigned to communities (or invited) by community administrators.
User group	Collects users for purposes that cut across the portal. User groups are defined by portal administrators.

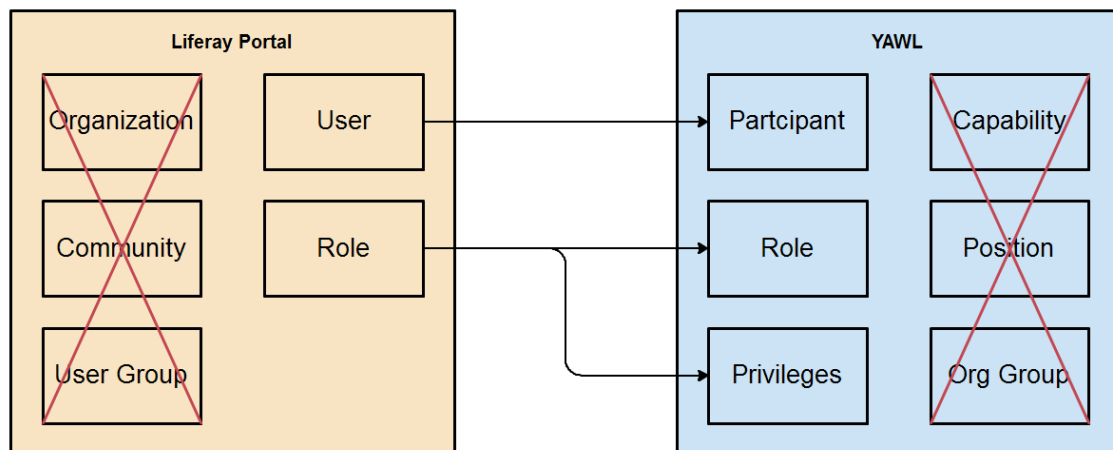


Figure 2.1.: Mapping of organizational data

2. The Examination Environment Components

Table 2.2.: YAWL's organizational model [HvdAAR10, cf. pp. 265 ff., 282]

Type	Description
Role	<p>“a duty or set of duties that are performed by one or more participants [...] a role may be considered as a grouping of participants who share the same (or similar) duties within an organization. [...] A role may be included in the distribution set for a task at design time, meaning that all of the participants performing that role (or any of its sub-roles) are to be considered as potential recipients of a work item created from the task at runtime” [HvdAAR10, p. 266].</p>
Capability	<p>“desired skill or ability that a participant may possess. [...] There may be several participants within an organization possessing the same capability, and a certain participant may possess a number of capabilities. In the YAWL model, a participant may possess zero or more capabilities. [...] may be included in a filter defined at design time” [HvdAAR10, p. 267].</p>
Position	<p>“typically refers to a unique job within an organization for the purposes of defining lines-of-reporting within the organizational model [...] a participant may hold zero or more positions. Importantly, a position may report to zero or one other positions [...] may be included in a filter defined at design time”[HvdAAR10, p. 267].</p>
Org Group	<p>“a functional grouping of positions [...] any grouping relevant to an organization. In the YAWL model, each position may belong to zero or one org groups. [...] may belong to a larger, more general org group [...] often also based on location [...] may be included in a filter defined at design time”[HvdAAR10, p.267]</p>
User Privileges	<p>“are granted by an administrator to a participant, on an individual basis [...] apply to the participant at all times, for all case instances - that is, they are privileges that are permanently owned by each individual participant”[HvdAAR10, p. 282]</p>

2. The Examination Environment Components

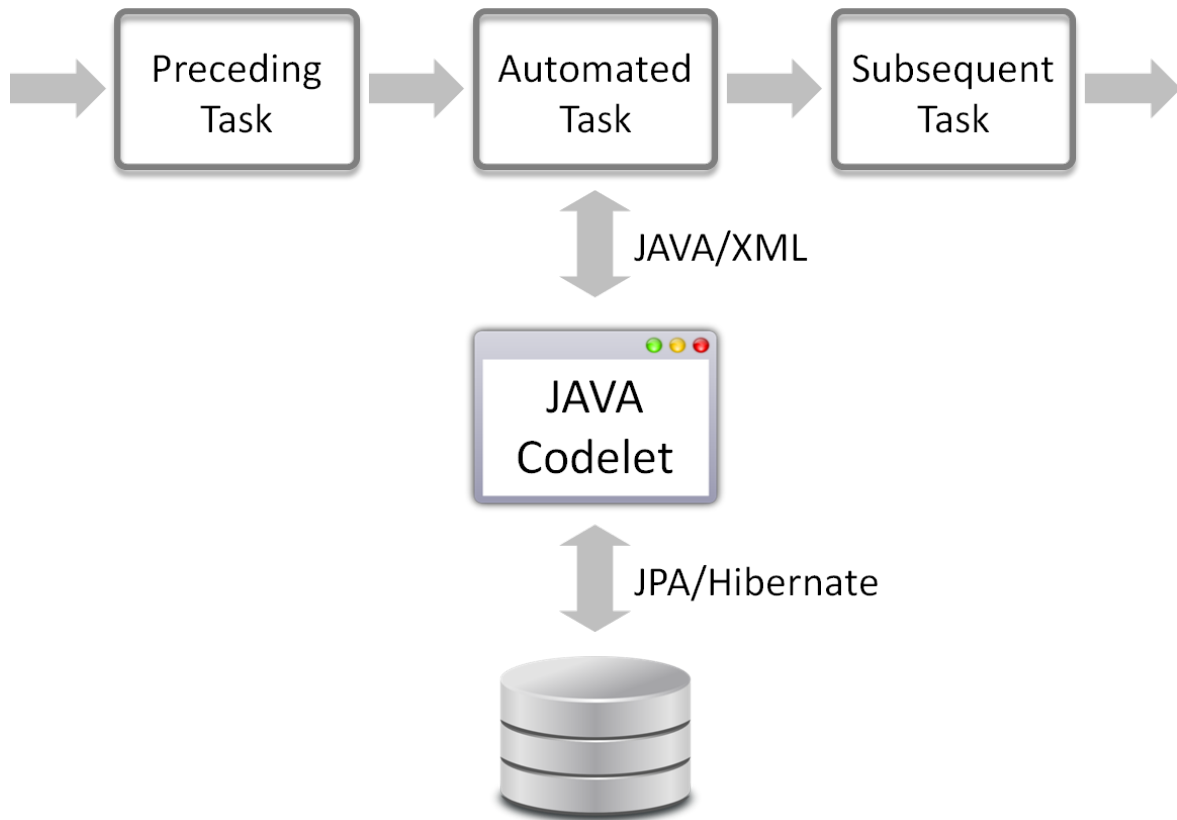


Figure 2.2.: Codelet operation schema

well working and flexible variant of interfaces between running workflows and external data or applications:

“Automated tasks are handled by codelet executions. A codelet is essentially a Java class that complies to a predefined Java interface, and is programmed against other predefined Java interfaces for controlling a work item and accessing work item data. [...] The codelet model offers the expressiveness and versatility of a programming language, and is therefore suitable for the implementation of operations that involve complex data manipulations” [HvdAAR10, p. 208].

This means a codelet can exchange all provided variables with the workflow task it is assigned to. As a Java class, it is able to access databases via respective existing frameworks. So, codelets should represent the way of choice regarding database access from inside a workflow.

2.5. Persistence via the Hyperjaxb3 Project

As previously described, the data types required for this integration approach are defined via XML schema definition (XSD), which is applicable in a YAWL workflow definition. In addition, the YAWL engine uses XML (or XML containing Java objects) for all matters of caching or exchanging data in a running workflow instance. Also, the data delivered to

2. The Examination Environment Components

the above-mentioned codelet is wrapped in such an XML structure which is not usable for relational persistence without significant limitations. As shown in figure 2.2, the data exchange between codelet and database shall be achieved using the Hibernate framework which is an implementation of the Java Persistence API (JPA). This framework uses object-relational mapping for persisting Java objects in databases. So, if a codelet obtains data as XML and needs Java objects for accessing the database, a conversion from one data format to the other is required. This can be done manually by fetching the elements out of the XML structure and mapping them to the respective attributes of a Java object. Because business objects (and the corresponding XML structures) can reach a high level of size and complexity, a manual mapping would be extremely expensive and error prone. Hence, another technology is needed.

The Hyperjaxb3 project bridges the gap between business objects in XML and Java. It provides relational persistence for JAXB objects by augmenting schema-derived classes with JPA annotations. Hyperjaxb3 is developed by a single author, A. Valikov, and has gained maturity within the last few years [Val09]. The core functionality of the Hyperjaxb3 project is a code generator which uses an XML schema definition to generate Java classes augmented with JPA and JAXB annotations.

The Java Architecture for XML Binding (JAXB) is an API which “simplifies access to an XML document from a Java program by presenting the XML document to the program in a Java format. The first step in this process is to bind the schema for the XML document into a set of Java classes that represents the schema” [OM03]. By using these classes, an XML document can be converted to Java objects (called *unmarshalling*), or Java object content can be converted (*marshalled*) to an XML structure or document [OM03, cf.].

Figure 2.3 shows the application of the Hyperjaxb3 project during build time and runtime. Starting point is an XML schema definition of the required data type or business object. Based on this definition, Java classes are generated which represent both JAXB-objects and JPA-objects (i.e., entity classes). At runtime, an incoming XML structure or document is unmarshalled [OM03, cf.] (i.e., converted) to a Java object corresponding to the respective class and written to the database via JPA. Obtaining data from the database works the other way round by converting a JPA-populated object to its XML counterpart.

In the context of a practical project, the author of this thesis developed a first YAWL workflow persistence layer based on Hyperjaxb3 in summer 2013, and this technology has been examined and successfully applied in two current master theses [Mor14, Dam14]. Hence, it provides a well working and flexible mechanism to connect the YAWL workflow interfaces (i.e., the codelets) to a JPA-based persistence layer and will come to use in this examination. The corresponding data access objects (i.e., the classes using conversion and entity classes for database access) depend on business context and on the methods needed by the overlying applications. Therefore, they are implemented manually.

2.6. Liferay Portal

As mentioned before, the YAWL WMS is open source and so the portal application used in this elaboration should also be. In addition, a comprehensive support and a high level of

2. The Examination Environment Components

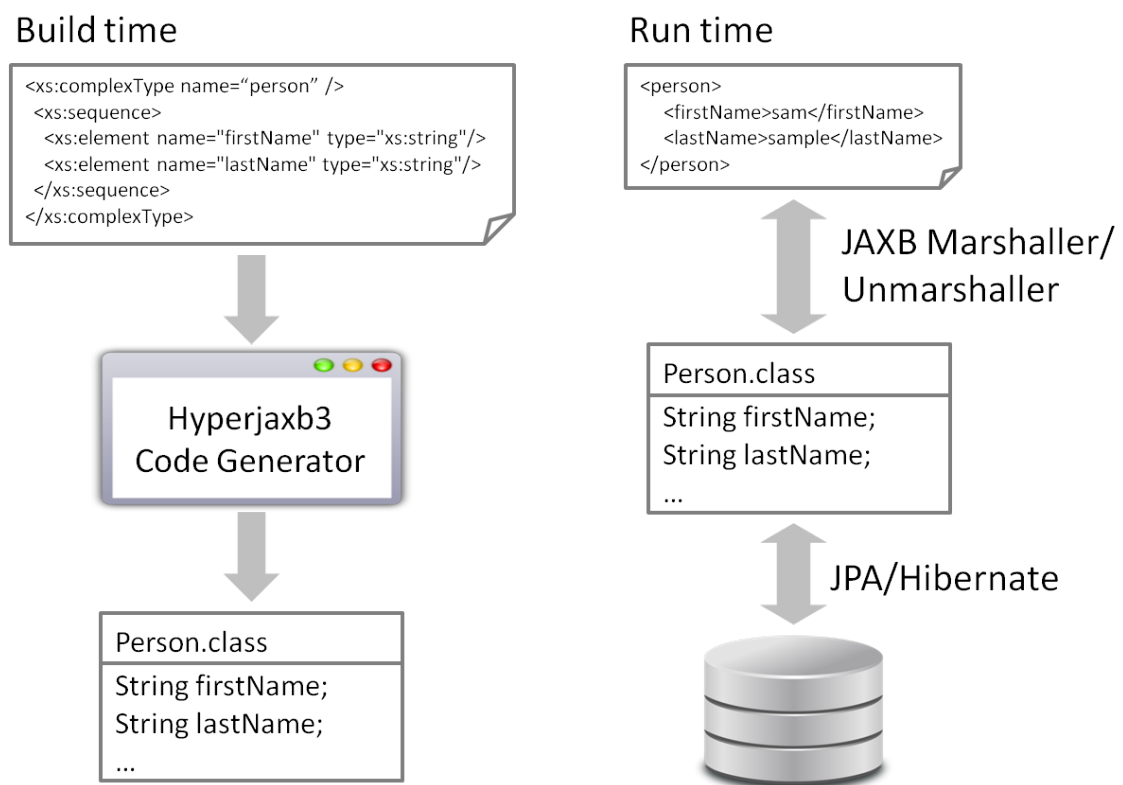


Figure 2.3.: Hyperjaxb3 operation schema

2. The Examination Environment Components

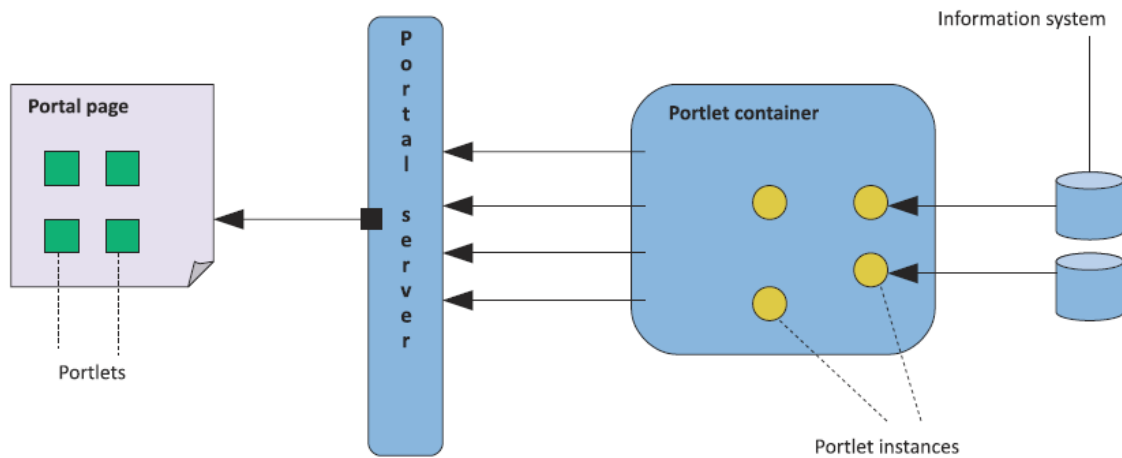


Figure 2.4.: Portal infrastructure [Sar12, p. 16]

maturity should be provided. Furthermore, the author of this thesis has worked with Liferay Portal before and gained experience with this application.

2.7. Portlet Applications with Vaadin

For the integration of YAWL into Liferay Portal, it is necessary to implement the respective applications for accessing YAWL services and workflow-collected data. As already mentioned, web pages or applications running in a portal are called portlets. Sarin defines a portlet as “a pluggable user interface component that provides specific content, which could be a service or information from existing information systems” [Sar12, p. 10].

“Portlets provide the user interface of the portal by accessing distinct applications, systems, or data sources and generating markup fragments to present their content to portal users. Some examples of portlets are a Weather portlet that provides weather information for a city by accessing a Yahoo! Weather RSS feed, or a Help Desk portlet that displays the pending help desk tickets from a database” [Sar12, p. 10].

Figure 2.4 shows the infrastructure of a portal with contained portlets. A portlet container manages different instances of portlet applications which themselves contain some application logic or data representation. On the portal page provided by the portal server, exactly one instance is displayed, i.e., every user session has its own instance of the portlet application.

2.7.1. Anatomy of a Portlet Application

Usually, a portlet application consists of at least three components. First is the Java implementation, which represents the back end and contains the business model as well as data processing and database access if required. The second mandatory component is

2. The Examination Environment Components

the portlet configuration consisting of several XML files. These provide the information the portal needs to handle the portlet application (e.g. how to display and which views are available). The third component represents the front end, i.e., the user interface. It usually contains components which are known from other web applications (e.g. graphics, styles, Java script, web pages).

A very usual way to create portlets is to implement them by using the Java server pages (JSP) technology. This means, the user interface is represented by an (X)HTML web page with nested elements for data representation or interaction (e.g. text fields or buttons). These elements are bound to variables or methods in separate Java classes, the so-called beans. These can either handle all the application logic or only a part of it, delegating functions and data processing to the back end. So, the JSP way of building portlet applications needs to bridge the gap between different technologies. Apart from the fact that the flexibility of the implementation depends on the flexibility of the used framework and its possible element-object bindings, a developer has to learn and to handle at least two different programming languages.

2.7.2. The Vaadin Framework

In contrast, a desktop application can be almost completely implemented using one technology. In Java, user interfaces are built by using the respective toolkits (e.g. AWT or Swing), which are also handled in the common object-oriented way of Java programming. There are no changes between different technologies, and having good Java skills is enough for a developer to write such an application. The Vaadin framework could be described as exactly such a toolkit, but for web application development.

“Vaadin is an AJAX web application development framework that enables developers to build high-quality user interfaces with Java, both on the server- and client-side. It provides a set of libraries of ready-to-use user interface components and a clean framework for creating your own components. The focus is on ease of-use, re-usability, extensibility, and meeting the requirements of large enterprise applications”[Grö13].

Figure 2.5 illustrates the architecture of a Vaadin-built web application. It consists of a server-side framework and a client-side engine which runs as Java script code. This engine manages interaction with the server and renders the user interface, whose logic runs as a servlet in the application server. The Vaadin framework bases on Google web toolkit (GWT) which provides a Java to Java script compiler and a comprehensive browser support. Because a Vaadin application can be seen as almost pure Java from the development point of view, nearly the whole application can be implemented in this language. Apart from Java script, which is a standard browser component, no plugins are required. In addition to the standard libraries contained in the Vaadin package, there is a large amount of add-ons available [Grö13, cf. p. 24].

“Vaadin supports running UIs as portlets in a portal, as defined in the JSR-286 (Java Portlet API 2.0) standard. The portlet UI is defined just as a regular UI, but deploying to a portal is somewhat different from deployment of regular web applications, requiring special portlet descriptors, etc. [...] In addition to providing user interface through the Vaadin UI, portlets can integrate with the portal

2. The Examination Environment Components

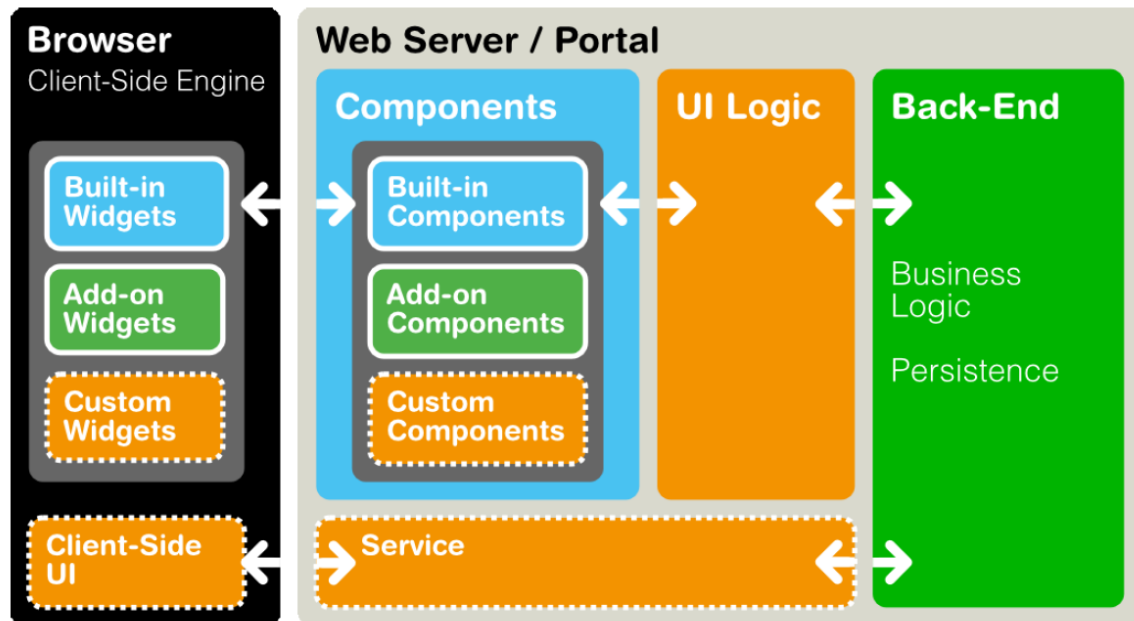


Figure 2.5.: Vaadin architecture [Grö13, p. 24]

to switch between portlet modes and process special portal requests, such as actions and events. [...] While providing generic support for all portals implementing the standard, Vaadin especially supports the Liferay portal” [Grö13, p. 362].

That means, a Vaadin-made web application front end, denoted as user interface (UI), can be run in a portal (i.e., as a portlet) without major changes in the way of implementation. In addition, a Vaadin portlet will be able to interact with the portal environment just as well as the above-mentioned JSP-based portlets. So, from the viewpoint of the author of this thesis as a good-skilled Java developer, the advantages of Vaadin outweigh. Hence, this framework is selected for the implementation of the portlets which are required for the integration of the YAWL WMS into Liferay Portal.

2.8. Overall Architecture

Table 2.3. provides an overview of the technologies discussed and selected in this chapter. These components are summarized in table 2.3. The YAWL WMS and the portal are deployed to an Apache Tomcat web application server. A sample data type of medium complexity is defined as an XML schema and represents the complex data type for the YAWL workflow definition and the basis for the entity classes which have to be generated by the Hyperjaxb3 project. The organizational data is exclusively managed by the portal and imported to the WMS by implementing an extension for YAWL’s Resource Service and mapping the necessary components from one organizational model to the other. The data, which is collected during workflow processing, is exchanged with the database by means of respective codelet applications. The Hyperjaxb3 project is applied to create a persistence layer which

2. The Examination Environment Components

Table 2.3.: Examination environment components summary

Type	Description
Application server	Apache Tomcat web application server, version pretended by server bundled with Liferay Portal
Business object / data types	XML schema definition
Organizational data	share of portal organizational data with WMS by implementing extension for YAWL Resource Service and mapping Liferay's organizational model to the one of YAWL
Workflow data interface	YAWL codelets accessing the database
Persistence for XSD data types	Hyperjaxb3 project for conversion of XML to instances of generated entity classes and back
Portal Environment	Liferay Portal at latest version
Framework for portlet applications	Vaadin web application development framework

can be used for data represented by Java objects and XML. Liferay Portal serves as the business portal environment in this examination, and the Vaadin web application development framework is used for the creation of required portlet application.

2.8.1. Excursus: Technical Architecture Modeling

Before continuing with an overview of the expected architecture, it is necessary to introduce the modeling standard used for the architecture diagrams in this thesis. Technical Architecture Modeling (TAM) is a standard which has been developed by SAP for internal purposes. The current version bases on the UML 2.0 standard, but extends the UML meta model by the FMC block diagrams which have been used at SAP since 1990. In contrast to architecture diagrams in UML, the TAM/FMC block diagram has a significantly less complex notation[Gro08, cf.].

As shown in figure 2.6, the notation of the TAM/FMC block diagram defines four types of components. As active components, agents (rectangular shapes) process information, read and write data from and to storages, and communicate with other agents via channels. Agents can be hardware systems, software applications, or human participants. Storages (rounded shapes) are passive components which keep information, are read and modified by agents, and can be used to exchange information. Channels (small circles on connecting arcs between agents) which transport but do not store information, are also passive. They

2. The Examination Environment Components

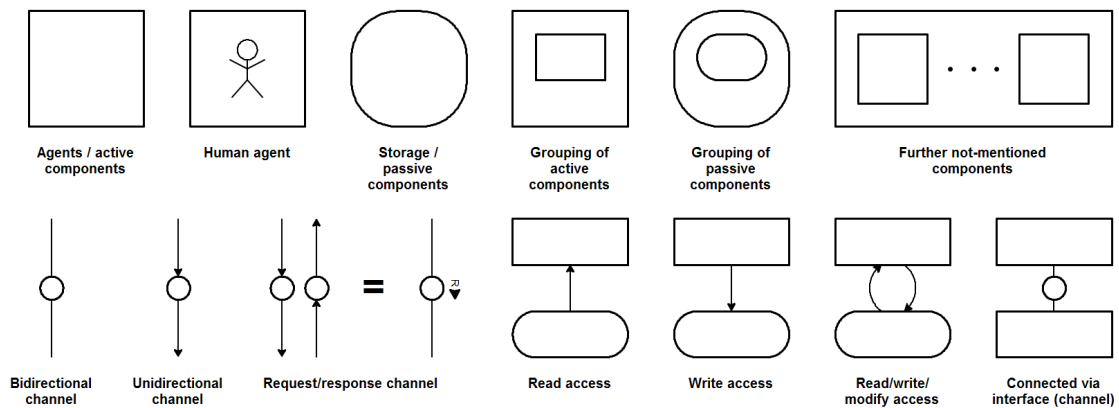


Figure 2.6.: TAM/FMC block diagram components

are used for communication between agents and can be unidirectional or bidirectional. Access arcs (directed arcs between agents and storages) indicate the direction of data flow [Boe13, cf. pp. 13-19].

2.8.2. Expected overall architecture

As shown in figure 2.7, the architecture of the integration approach is expected to consist of five main components. Liferay Portal (red) and the YAWL WMS (blue) already exist and need to be customized and adjusted. The other components have to be implemented and are aggregated to three main components according to their application context.

The first application context is represented by the extensions for YAWL, which represent customizing according to the requirements. So, one subcomponent is the organizational data extension which replaces the respective class in the Resource Service. It is connected to the service APIs of Liferay from which it fetches the users and roles to map this data to the organizational model of YAWL. The codelet interfaces also belong to this context because they extend the existing YAWL codelet library and have to be deployed to a specific package inside the Resource Service, which also calls the codelets that are assigned to a running workflow instance. Because the fetched and mapped orgdata is also transported to Resource Service, the entire YAWL extension component (and so all of its subcomponents) has a bidirectional channel to the Resource Service. In addition, the codelet interfaces have a channel to the persistence layer main component for database access.

The second main component to be implemented is the persistence layer, which contains all required classes and configuration files for the access to the business database. As already mentioned, the entity classes are generated by the Hyperjaxb3 project, whose runtime environment (required for object-XML conversion) is also part of the component. As the only subcomponent communicating with external components, the data access objects (DAOs) have bidirectional channels to the business application portlets (i.e., to the portlets which allow to display or modify workflow collected data in the portal) and to the codelet interfaces which are responsible to access the database from within a running

2. The Examination Environment Components

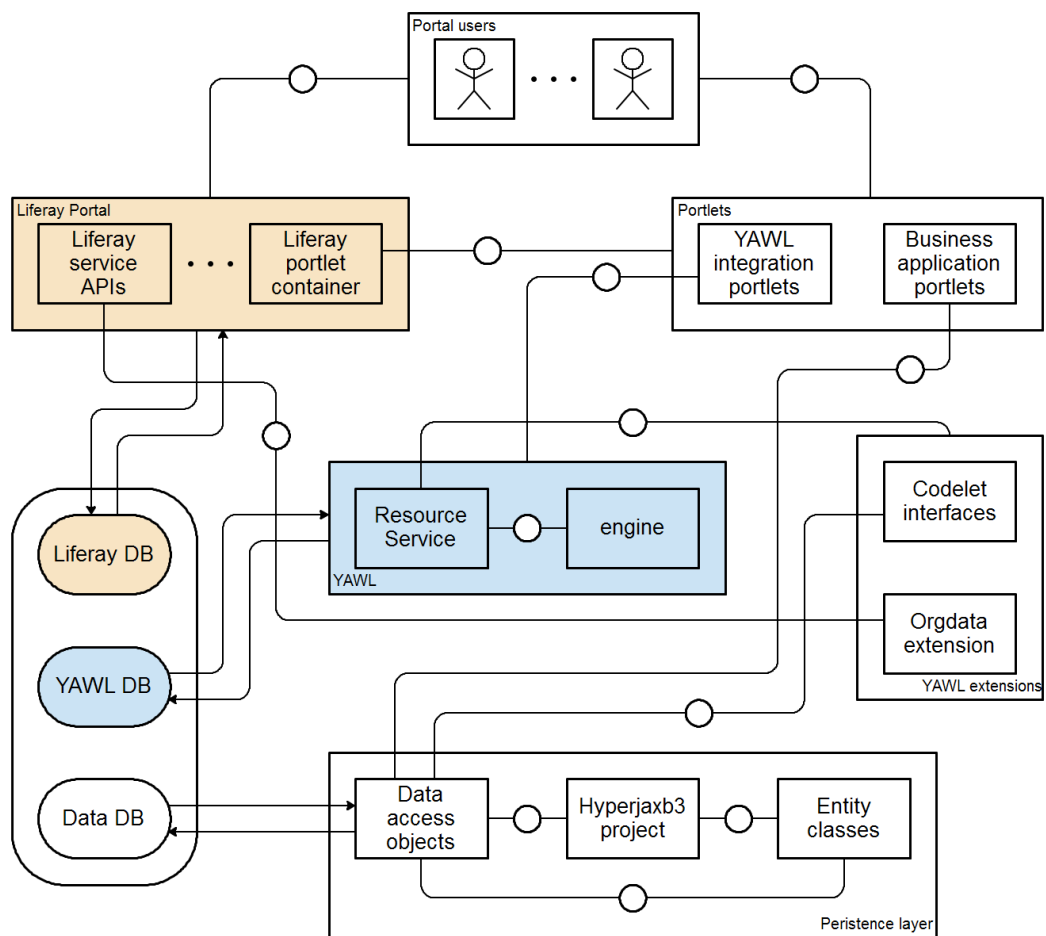


Figure 2.7.: Expected overall architecture

3. Integration Approach

workflow instance. Because component-internal communication takes place between all shown subcomponents (e.g., the entity classes are used by the data access objects for reading from or writing to the database as well as by Hyperjaxb3 for conversion), they are all connected to each other via bidirectional channels. As can be seen, the DAOs are the only component in the entire architecture which has modifying access to the database where the business objects are persisted.

The last main component of the draft architecture is to describe. It is represented by the applications running in the portal (i.e., the portlets) which can be split into two categories. The YAWL integration portlets interact with the YAWL WMS and provide information about running workflow instances (e.g., assigned tasks). They also allow the respective user to process assigned workitems (e.g., to fill out a form) and are only connected to the YAWL main component. The second category includes the business application portlets which allow users to work on the business objects that have been collected or shall be completed by workflows. Because all portlets run inside Liferay and communicate with the portal environment, the entire component has a bidirectional channel to the portlet container.

The portal users access the portal environment and the contained portlets using their browsers. Although portal and portlets are different main components in this architecture, no difference is visible from the user perspective at runtime. The portlets application are displayed as sub-pages inside the portal pages. Both the YAWL extensions component and the persistence layer work completely in the background and are therefore not visible at all at runtime. While the portal database stores application states, configurations, organizational data, and content (e.g., web pages), the YAWL database only contains workflow states, current variable values, workflow specifications, and other workflow-relevant data. All business data is stored in a separate database.

3. Integration Approach

This chapter describes the development of the integration components discussed in the previous chapter. Essential code and configurations are provided and elucidated. The complete source code and all necessary files are provided on the data carrier attached to this thesis and as download. Relevant URLs for downloading source code or applications can be found in the appendix and are therefore not specified here. It should be noted that the source code is written in a way that makes it easy to read and to comprehend. Therefore, some awkward expressions and code redundancies are accepted here but should be eliminated in a derivation for productive use.

3.1. Installation of Liferay Portal and YAWL on one Application Server

As already mentioned in the previous chapter, Liferay Portal and the YAWL WMS shall be installed on the same web application server. Because the goal of this integration approach is to provide a properly implemented application sample which allows to be reconstructed and to derive usable models for different application contexts, the way of copying the two Tomcat bundles into each other (as done in [Ros13, cf. p. 31 f.]) is not appropriate.

Liferay Portal is set up as root application on the Apache Tomcat application server which is used in this examination. This means that the inbuilt root application for managing the server is replaced by Liferay and several configuration files are changed. This is not required for the installation of the YAWL WMS, which is represented by a few single web applications. These are able to run on the server with a minimal configuration effort. Because Liferay Portal is a well documented and comprehensively supported software, the installation of this portal on a web server from scratch is not a problem. It is not necessary because a ready-to-use Tomcat bundle is available on the Liferay download page. This bundle already contains all configurations which would have to be done during a manual installation. Although Liferay takes full control on the web server and establishes its own deployment mechanism (which is only usable for portlet deployment), the original application management of Apache Tomcat is still active and working without limitations apart from a missing web application management page. So, a standard web application, in the following also denoted as webapp, can be deployed and run the usual way (i.e., by copying it packed or unpacked to the Tomcat webapps folder and restarting the server). Hence, the Liferay Portal Tomcat bundle is applied as basis of the examination environment. In addition, the required YAWL WMS webapps are installed using the deploy mechanism of the underlying Tomcat.

3.1.1. Liferay Portal

The examination environment is set up on a virtual machine running Debian Linux (version 3.2.54-2) with Java runtime environment (version 1.7.0) and PostgreSQL (version 9.1). Three database users are set up. These are liferay, yawl, and personnel (representing the sample business database). Corresponding to the users, three databases are created with the same names. For developing purposes, the passwords are equal to the names of users and databases.

The current Liferay Portal Tomcat bundle (version 6.2-ce-ga2) is downloaded from the Liferay download page as zip archive and unpacked in the */opt* folder of the Linux installation as */opt/lr622yawl3*. One way of setting up the portal would be to just start it and run through a setup routine. Instead of this, the basic configuration is done before the first start by creating a *portal-ext.properties* file in the root folder (i.e., */opt/lr622yawl3*) of the portal.

```
#
# PostgreSQL Settings
#
jdbc.default.driverClassName=org.postgresql.Driver
jdbc.default.url=jdbc:postgresql://localhost:5432/liferay
```

3. Integration Approach

```
jdbc.default.username=liferay
jdbc.default.password=liferay
#
# Disable Liferay Auto-Setup
#
setup.wizard.enabled=false
#
# Change certain default settings
#
default.admin.password=password
default.admin.screen.name=christian
default.admin.email.address.prefix=christian
default.admin.first.name=christian
default.admin.middle.name=
default.admin.last.name=freihoff
company.default.web.id=freihoff.org
company.default.name=Thesis Demonstrator
company.default.locale=en_US
company.login.prepopulate.domain=true
look.and.feel.modifiable=true
terms.of.use.required=false
company.security.auth.type=screenName
#
# Disable users private/public pages
#
layout.user.private.layouts.auto.create=false
layout.user.public.layouts.auto.create=false
layout.user.private.layouts.enabled=false
layout.user.public.layouts.enabled=false
#
# Disable some user management settings
#
users.reminder.queries.enabled=false
users.reminder.queries.required=false
users.reminder.queries.custom.question.enabled=false
users.email.address.required=false
# Portal-wide widget set
vaadin.widgetset=com.vaadin.portal.gwt.PortalDefaultWidgetSet
# Theme to use
vaadin.theme=liferay
```

Listing 3.1: portal-ext.properties

As can be seen in listing 3.1, the database settings are configured corresponding to the previously created database. Because the above-mentioned setup routine is circumvented, it has to be disabled. By default, Liferay would create a basic user account with administrative rights and assign default values to company and site name. This is overridden, as well as several other values relevant for user and authentication management. Because the Vaadin web application development framework is used for the portlet applications, some

3. Integration Approach

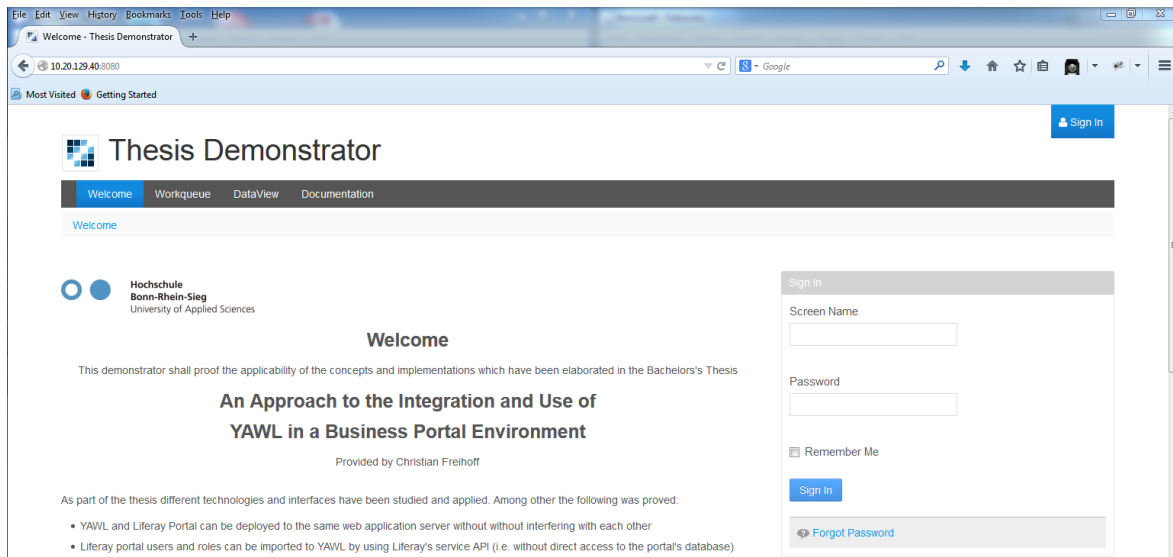


Figure 3.1.: Liferay welcome page

required settings are also provided. The applied properties are taken from [lif14] where a large number of additional settings can be found.

By default, the memory which can be assigned to the running portal instance is set to a very low value. To prevent memory problems at runtime, the file *setenv.sh* in the directory */opt/lr622yawl3/tomcat/bin* needs to be changed. As shown in listing 3.2, the maximum memory value *-Xmx* is changed to 4096 megabytes, which should be enough for this purpose. In addition, the current timezone is set to *-Duser.timezone=Europe/Berlin* corresponding to the location of the demonstrator server. When the portal shall be run on a windows machine, the file *setenv.bat* needs to be changed instead.

```
CATALINA_OPTS="$CATALINA_OPTS -Dfile.encoding=UTF8 -Djava.net.  
preferIPv4Stack=true -Dorg.apache.catalina.loader.WebappClassLoader.  
ENABLE_CLEAR_REFERENCES=false -Duser.timezone=Europe/Berlin -Xmx4096m -XX  
:MaxPermSize=384m"
```

Listing 3.2: setenv.sh

After applying these settings, the portal starts up by calling */opt/lr622yawl3/tomcat/bin/startup.sh* (or *startup.bat* on windows). Because Liferay Portal is a comparatively huge application, the startup takes a few minutes. After that, it is accessible on port 8080 at the respective server URL (e.g., *http://10.20.129.40:8080*).

Figure 3.1 shows the welcome page of the finished demonstrator portal enriched with some HTML content. The login portlet at the right demands for the screen name which corresponds to the setting made in the *portal-ext.properties* above. The site name (set by the company default name) is displayed correctly, and a login using the defined credentials is successful.

3. Integration Approach

3.1.2. YAWL WMS

As discussed above, the WMS will be installed on the existing web application server of the Liferay Tomcat bundle. Therefore, the YAWL core services package at version 3 is obtained from the respective download pages. It contains seven web applications packed as WAR files (*documentStore*, *mailService*, *monitorService*, *resourceService*, *workletService*, *yawl*, *yawlWSInvoker*) from which the relevant ones for this examination are *yawl* (i.e., the workflow engine) and *resourceService*. Because YAWL is a system in ongoing development, the core services in this package is augmented by the available latest builds (i.e., applications containing the latest bugfixes) of some core services. At the time of this integration approach, there are improved versions of *workletService*, *resourceService* and *textityawl* available. It should be mentioned that several bug fixes, and so some of the latest builds, are results of this examination. More details about this aspect are discussed later at the end of this chapter.

Basically, the YAWL core services are ready to use and can be deployed to a web application server immediately. Only one configuration needs to be adjusted. Depending on the used versions, some of the core services contain a *hibernate.properties* file in the */WEB-INF* folder. At the current version including the latest builds, these are *documentStore*, *resourceService* and *yawl*. Ignoring the latest builds, the *workletService* also contains such a configuration file. The YAWL core services are already configured to use a database named *yawl* at a PostgreSQL database system which corresponds with the basic configuration of the demonstrator server. As described above, user name and password for the YAWL database are *yawl* and *yawl*. These two settings have to be changed in all existing *hibernate.properties* files as shown in listing 3.3.

```
...

#####
### Platforms ###
#####

## PostgreSQL

hibernate.dialect org.hibernate.dialect.PostgreSQLDialect
hibernate.connection.driver_class org.postgresql.Driver
hibernate.connection.url jdbc:postgresql:yawl
hibernate.connection.username yawl
hibernate.connection.password yawl

...
```

Listing 3.3: hibernate.properties

After applying the configuration changes, the Liferay-bundled Tomcat server is stopped and the YAWL core services are deployed. This can be done by copying either the packed WAR files or the unpacked web application folders to */opt/lr622yawl3/apache-tomcat-[/...]/webapps*. Here the second alternative is chosen and the web applications are copied as folders into the above-mentioned folder inside the Liferay Tomcat bundle (cf. figure 3.2).

3. Integration Approach

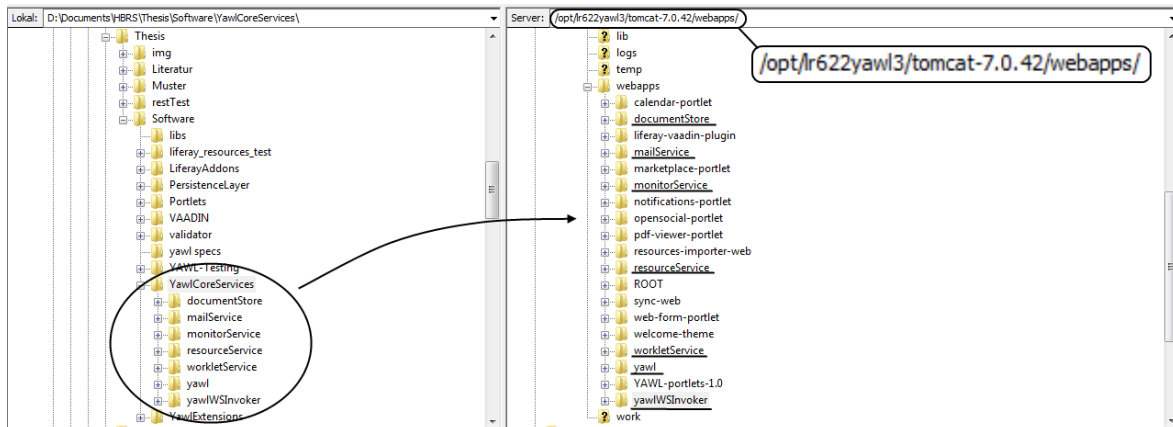


Figure 3.2.: Deployment of YAWL core services

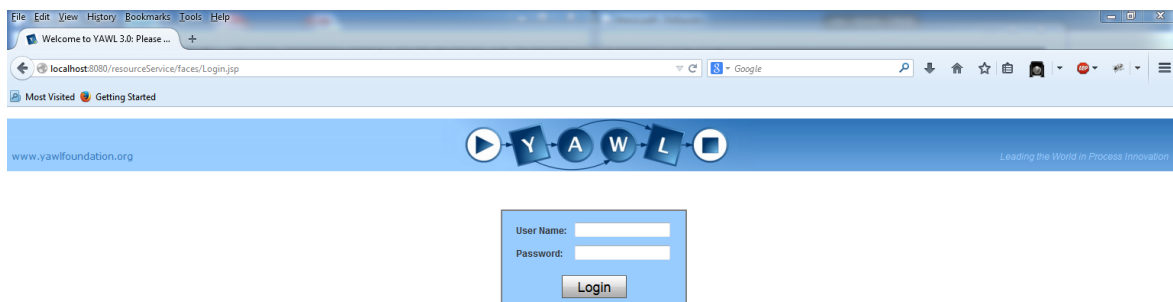


Figure 3.3.: YAWL Resource Service login

After starting the Tomcat server, all deployed YAWL applications can be accessed by appending the folder name of the web application to the server address, including the port as explained above (e.g., <http://10.20.129.40:8080/resourceService> for the Resource Service application, see figure 3.3).

3.1.3. Additional Dependencies and Conclusion

For later steps in this integration approach, the YAWL Resource Service requires some additional Java dependencies which are mentioned at this point. These are the *hyperjaxb3-ejb-runtime*, the *jaxb2-basics-runtime* and the *hibernate-entitymanager*. Current versions, which correspond to the versions of related software components, are added to the `/opt/ir622yawl3/tomcat-7.0.42/webapps/resourceService/WEB-INF/lib` folder of Resource Service. The YAWL core services provided on the attached data carrier and as download already contain all required files and configurations.

The way of installing Liferay and YAWL on one application server by adding the YAWL core services to the Liferay Tomcat bundle has been repeatedly tested thoroughly without detecting any problems or limitations and can therefore be considered as absolutely proper and easily reproducible.

3.2. Definition of Sample Workflow and Data Type

To use a well-known example, the business process is one which is applied in most companies and organizations: the check in of a new employee. Such a procedure may be complicated when the respective person has to pass several stations in a fixed order. Before starting the workflow design, the sample organizational structure and the sample business object are defined.

As shown in table 3.1, apart from the new employee who has to be checked-in, there are four relevant roles in this process. The HR-Administrator is responsible for all personnel matters and shall start the process. This role also has to check the already collected basic data and must pass the automatically generated login data to the new employee. The System-Administrator has to create company email address and office phone number for his new colleague, and the Warehouse-Manager hands out required items. In this process, these are an office key and a card for time recording. Because the financial accountant has to ensure that the employee receives his salary, he checks and validates the bank account data which the new colleague has declared before.

Table 3.1.: Sample organizational structure

Name	Role	Job in sample process
Paul Personnel	HR-Administrator	Creates new employee's dataset, starts check-in process and provides login data to employee
Alvin Administrator	System-Administrator	Creates new eMail and phone account for his new colleague
Fred Finance	Financial-Accountant	Checks and validates the bank account data provided by new employee
Walter Warehouse	Warehouse-Manager	Hands out office key and time card to the new colleague

The sample business object (i.e., the new employee) is shown in listing 3.4. The *employeeType* contains several string elements representing names, dates, and other primitive values. To prove that YAWL and Hyperjaxb3 work well with more complex data types, there is also an enumeration with three selectable values for the gender and some complex types containing additional elements. Finally, the employee business object has the following attributes:

- First name (mandatory)
- Last name (mandatory)
- Date of birth (mandatory)
- Nationality
- Gender (male, female or undefined)

3. Integration Approach

- Private contacts (email, phone, mobile)
- Address (street, number, postcode, city, country)
- Bank account (IBAN, BIC)
- Handed out items (office key, time card)
- Official contacts (company email, office phone)

The elements *blocked* and *workflowStatus* are no attributes of the employee, but they are required for the application. As discussed, the database shall be equally accessed by YAWL workflows and portlet applications. Although running on the same machine and web application server, the two systems (i.e., WMS and portal) are independent from each other and do not share a persistence context. The WMS is intended to load a specific data set at the beginning of a process and to update the database when the workflow has finished. Depending in the workflow specification, this may take several minutes, hours, or even days. So, if a data set which is worked on in a running workflow instance is manipulated (e.g., an employee's phone number is changed) by using the business application in the portal, these changes would be overwritten when the workflow finishes. To allow the different applications to be aware of each other working on a data set, a simple boolean *blocked* flag is added to the business object. When a workflow instance is started and an object is fetched from the database, this flag is set to *true*, and the business application's access to this data set is restricted to read only as long as the process is running.

The *workflowStatus* shall help the portlet application to offer the right actions for the respective data set. For example, if an employee has already been checked in, the action for starting the check-in workflow shall not be offered anymore. The value of this variable shall be set by the application which is currently working on the business object. When creating a new employee data set, three variables are mandatory to provide distinguishability. An employee whose name is John Smith might not be unique in a medium-sized or even small company when only first and last name are mandatory attributes, but having two persons with the same name and the same date of birth is rather less probable. Of course, personnel numbers, which are common in productive use, would be another good option.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jaxb="http://
  java.sun.com/xml/ns/jaxb" jaxb:version="1.0">

  <xs:annotation>
    <xs:appinfo>
      <jaxb:globalBindings generateIsSetMethod="true"/>
      <jaxb:schemaBindings>
        <jaxb:package name="org.freihoff.yawl.persistence.generated"/>
      </jaxb:schemaBindings>
    </xs:appinfo>
  </xs:annotation>
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Employee schema for Thesis Demonstrator.
      Copyright 2014 Christian Freihoff
    </xs:documentation>
  </xs:annotation>
```

3. Integration Approach

```
</xs:documentation>
</xs:annotation>

<xs:element name="employee" type="employeeType"/>

<xs:complexType name="employeeType">
  <xs:sequence>
    <xs:element name="firstName" type="xs:string"/>
    <xs:element name="lastName" type="xs:string"/>
    <xs:element name="dateOfBirth" type="xs:date"/>
    <xs:element name="nationality" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
    <!-- following elements defined below-->
    <xs:element name="gender" type="genderType" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="privateContacts" type="privateContactsType"
      minOccurs="0" maxOccurs="1"/>
    <xs:element name="address" type="addressType" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="bankAccount" type="bankAccountType" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="handedOutItems" type="handedOutItemsType"
      minOccurs="0" maxOccurs="1"/>
    <xs:element name="officialContacts" type="officialContactsType"
      minOccurs="0" maxOccurs="1"/>
    <xs:element name="blocked" type="xs:boolean" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="workflowStatus" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="addressType">
  <xs:sequence>
    <xs:element name="street" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="number" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="postcode" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
    <xs:element name="city" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="country" type="xs:string" minOccurs="0"
      maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="bankAccountType">
  <xs:sequence>
    <xs:element name="IBAN" type="xs:string" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
```

3. Integration Approach

```
        "1"/>
        <xs:element name="BIC" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="genderType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="male"/>
        <xs:enumeration value="female"/>
        <xs:enumeration value="undefined"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="privateContactsType">
    <xs:sequence>
        <xs:element name="eMail" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="phone" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="mobile" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="handedOutItemsType">
    <xs:sequence>
        <xs:element name="officeKey" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="timeRecordingCard" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="officialContactsType">
    <xs:sequence>
        <xs:element name="phone" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="eMail" type="xs:string" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

Listing 3.4: employee.xsd

For creating the workflow definition, YAWL provides a convenient graphical editor. Because there are still several bugs to be expected in the current version 3, the matured and relatively well-working version 2.3.5 is used. The XSD data type defined above is inserted into the workflow definition and is available during design time the same way as the data types which YAWL provides by default. Figure 3.4 shows a graphical representation of the finished workflow definition

3. Integration Approach

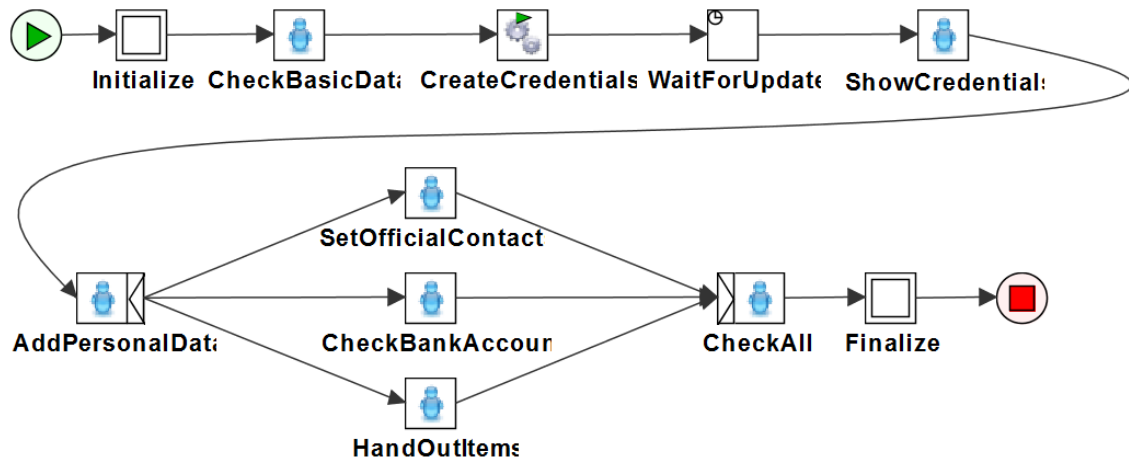


Figure 3.4.: Sample workflow

The workflow shown in figure 3.4 shall first load the new employee's dataset from the database (task *Initialize*). In the task *CheckBasicData* the HR-Administrator checks the basic data after its correctness. After that, an automated task (*CreateCredentials*) generates user credentials for the new employee and adds a new account in the system (i.e., in the portal). The organizational data of the portal is shared to the WMS in regular time intervals, so the *WaitForUpdate* timer task does nothing other than pausing the workflow for exactly the length of this interval. Afterwards, the generated credentials are shown to the HR-Administrator (*ShowCredentials*). He passes them to the new employee, who has to fill out a detailed form by himself (*AddPersonalData*). In the task *SetOfficialContacts* the System-Administrator adds an email address and a phone number to the data set. The Financial-Accountant checks the validity of the bank account data, inserted by the new employee himself, and makes corrections if necessary (*CheckBankAccount*). When the three last-mentioned tasks are finished, the *CheckAll* task allows the new employee to have a concluding look on his completed dataset.

3.3. Implementation of the Persistence Layer

For the persistence layer, three main components are required. First is the Hyperjaxb3 project itself, which can be downloaded from the project homepage. For this examination, the empty maven project is used. The project contains all dependencies and configurations which are necessary to generate the JAXB/JPA entities and related classes. The second important component is the XSD representation of the business object as discussed above and shown in listing 3.4.

3.3.1. Generation of Entity Classes

First step is to insert the XSD data type into the empty project and run a first build. The Hyperjaxb3 project generates the entity classes, an object factory class and a *persistence.xml* file. As can be seen in figure 3.5, the *employee.xsd* leads to the generation of several classes. The enumeration type and each complex type in the definition result in an

3. Integration Approach

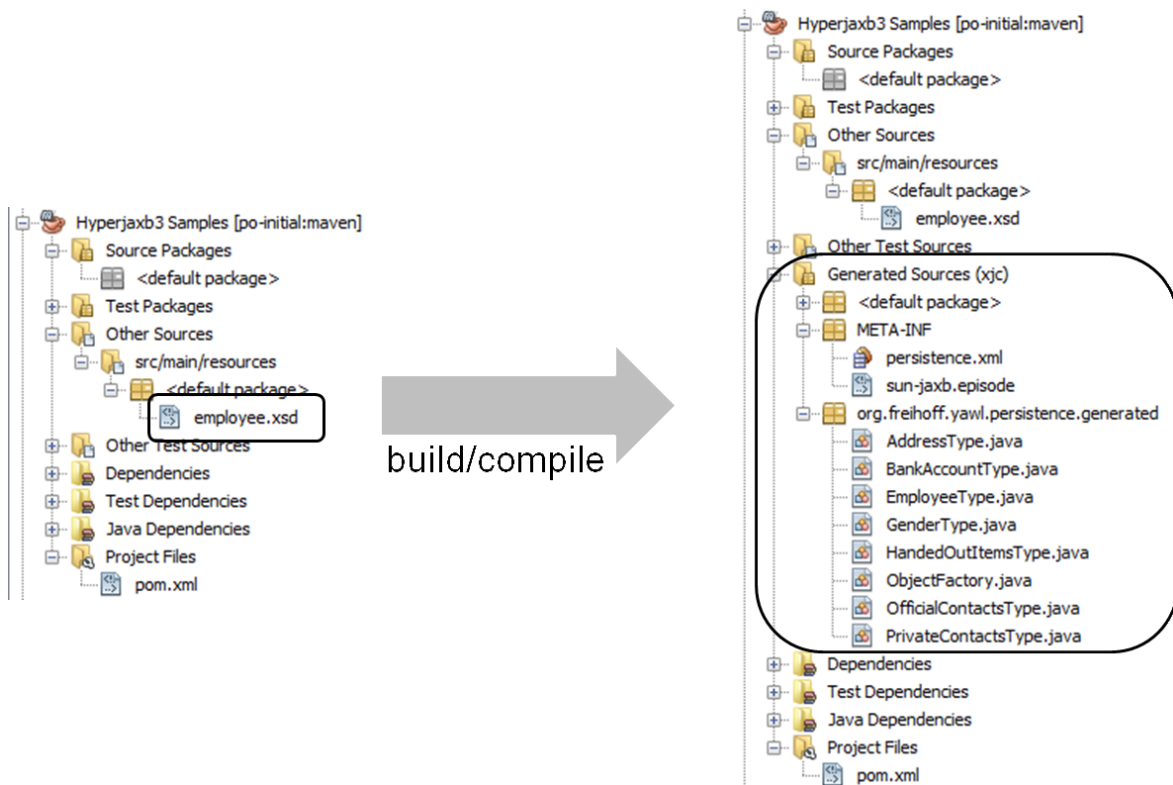


Figure 3.5.: Entity class generation from XSD

individual class. Listing 3.5 shows the *persistence.xml* which only represents the persistence unit corresponding to the generated entities and contains no database specifications. The object factory creates Java entity classes and JAXB elements as Java representation of XML content. Listing 3.6 shows the two different methods for creation of both object types.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence version="1.0" xsi:schemaLocation="http://java.sun.com/xml/ns/
  persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
  http://java.sun.com/xml/ns/persistence/orm http://java.sun.com/xml/ns/
  persistence/orm_1_0.xsd" xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:orm="http://java.sun.com/xml/ns/persistence/orm" xmlns:xsi="http://
  www.w3.org/2001/XMLSchema-instance">
  <persistence-unit name="org.freihoff.yawl.persistence.generated">
    <class>org.freihoff.yawl.persistence.generated.AddressType</class>
    <class>org.freihoff.yawl.persistence.generated.BankAccountType</class>
    <class>org.freihoff.yawl.persistence.generated.EmployeeType</class>
    <class>org.freihoff.yawl.persistence.generated.HandedOutItemsType</
    class>
    <class>org.freihoff.yawl.persistence.generated.OfficialContactsType</
    class>
    <class>org.freihoff.yawl.persistence.generated.PrivateContactsType</
```

3. Integration Approach

```
        class>
    </persistence-unit>
</persistence>
```

Listing 3.5: persistence.xml

```
...

    public EmployeeType createEmployeeType() {
        return new EmployeeType();
    }

...

    @XmlElementDecl(namespace = "", name = "employee")
    public JAXBElement<EmployeeType> createEmployee(EmployeeType value) {
        return new JAXBElement<EmployeeType>(_Employee_QNAME, EmployeeType.
            class, null, value);
    }

...
```

Listing 3.6: ObjectFactory.java

3.3.2. Database Configuration and Data Access Objects

After running the Hyperjaxb3 code generator, the required entity classes are ready for use, but are useless without the third main component of the persistence layer: the data access objects and the database-related settings. The data access objects (DAOs) and their methods depend on the business context, and the database connection settings depend on the used database. Hence, nothing speaks against implementing this part manually according to requirements and context.

The persistence layer shall provide basic CRUD (*Create, Read, Update, Delete*) functionality for both Java objects and XML data representation. Because the sample workflow shall only read and update existing business objects and is not intended to create or delete, the XML-related DAO can be implemented without the respective methods. On the other hand, these data access objects must contain methods for marshalling and unmarshalling (i.e., for converting object to XML and vice versa) and are therefore expected to be not less complex. The business application portlets are expected to require the full CRUD ability and some additional functions in accordance with the business context (e.g., a database search by an employee's name).

Persistence Properties

As mentioned above, the Hyperjaxb3 project generates the persistence unit without any configuration regarding the database access. This is caught up by creating a *persistence.properties* file which contains dialect, driver, address, user name, password and

3. Integration Approach

additional basic settings according to the existing database that has been set up at the beginning of this chapter (cf. listing 3.7).

```
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
hibernate.connection.driver_class=org.postgresql.Driver
hibernate.connection.username=personnel
hibernate.connection.password=personnel
hibernate.connection.url=jdbc:postgresql://localhost:5432/personnel
hibernate.cache.provider_class=org.hibernate.cache.HashtableCacheProvider
hibernate.jdbc.batch_size=0
hibernate.hbm2ddl.auto = update
```

Listing 3.7: persistence.properties

Java Object DAOs

The data access object for data represented by Java objects is implemented in a usual way. When instantiated, it loads the database context using the generated *persistence.xml* and the manually implemented *persistence.properties*. An excerpt of the *EmployeeAdapter.java* is shown in listing 3.8. The frequently occurring *hjid* variable represents the database primary key defined by the Hyperjaxb3 project.

In addition to the generally customary methods for CRUD operations, a method *setBlocker(Long hjid, boolean value)* is implemented which sets the above-discussed *blocked* variable in the employee business object. The database accessing methods use instances of the *EntityManager* (created by the *EntityManagerFactory*) with a short lifecycle. Transactions are carried out immediately and the objects are kept in the persistence context only as long as absolutely necessary to avoid problems with multiple accesses by different applications.

```
public class EmployeeAdapter {

    private EntityManagerFactory entityManagerFactory;

    public EmployeeAdapter() {
        setPersistenceProperties();
    }

    public Long persistEmployee(EmployeeType employee) {
        final EntityManager entityManager = entityManagerFactory.
            createEntityManager();
        entityManager.getTransaction().begin();
        entityManager.persist(employee);
        entityManager.getTransaction().commit();
        entityManager.close();
        return employee.getHjid();
    }
}
```

3. Integration Approach

```
...

public List<EmployeeType> getEmployeesByName(String name) {
    final EntityManager entityManager = entityManagerFactory.
        createEntityManager();
    List <EmployeeType> list = entityManager.createQuery("select e from
        EmployeeType e where lower(e.lastName) like :name or lower(e.
        firstName) like :name")
        .setParameter("name", "%"+name+"%")
        .getResultList();
    entityManager.close();
    return list;
}

...

public void setBlocker(Long hjid, boolean value){
    final EntityManager entityManager = entityManagerFactory.
        createEntityManager();
    EmployeeType employee = entityManager.find(EmployeeType.class, hjid);
    employee.setBlocked(value);
    entityManager.getTransaction().begin();
    entityManager.merge(employee);
    entityManager.getTransaction().commit();
    entityManager.close();
}

...

private void setPersistenceProperties() {
    final Properties persistenceProperties = new Properties();
    InputStream inputStream;
    inputStream = getClass().getClassLoader().getResourceAsStream("
        persistence.properties");
    try {
        persistenceProperties.load(inputStream);
    } ...
    entityManagerFactory = Persistence.createEntityManagerFactory("org.
        freihoff.yawl.persistence.generated", persistenceProperties);
}
}
```

Listing 3.8: EmployeeAdapter.java

3. Integration Approach

DAOs for XML-Represented Data

As mentioned above, additional steps are required for exchanging XML-carried data with the database. Because the database access is done by JPA (i.e., by the use of Java objects), data has to be converted from XML to Java and vice versa. While some methods (e.g., *setBlocker*, *delete*) in the *EmployeeXmlAdapter* are equal to the ones in the *EmployeeAdapter*, a special attention has to be paid to the methods for reading, writing, and updating.

As shown in listing 3.9, the method for writing (i.e., persisting) receives a string which shall contain an XML data representation. The string is converted (*unmarshalled*) to an object which is casted to a *JAXBElement* object, which in turn is mapped to an *EmployeeType* entity object. This entity finds its way into the database using the above-discussed *EntityManager*. Finally, the *hjid* (i.e., the primary key) of the persisted data set is returned.

While the method for updating is very similar, the method for reading data from the database goes the opposite way. It receives a primary key and fetches the respective object from the database using the *EntityManager*. This object is converted (*marshalled*) to a *JAXBElement* which is written to an XML-containing output stream. Finally, a string representation of this output stream is returned.

```
public class EmployeeXmlAdapter {

    ...

    public String read(Long hjid) {
        final EntityManager entityManager = entityManagerFactory.
            createEntityManager();
        EmployeeType employee = entityManager.find(EmployeeType.class, hjid);

        final Marshaller marshaller;
        OutputStream outputStream = new ByteArrayOutputStream();
        try {
            marshaller = context.createMarshaller();
            marshaller.marshal(objectFactory.createEmployee(employee),
                outputStream);
        } catch (JAXBException ex) {
            LOG.error("JAXBException while marshalling");
        }
        String outputString = outputStream.toString();

        entityManager.close();

        return outputString;
    }

    public Long write(String employeeString) {
        final Object object;
```

3. Integration Approach

```
final EmployeeType employee;
Long hjid = null;
try {
    final Unmarshaller unmarshaller = context.createUnmarshaller();
    InputStream inputStream = new ByteArrayInputStream(employeeString
        .getBytes("UTF-8"));
    object = unmarshaller.unmarshal(inputStream);
    employee = ((JAXBElement<EmployeeType>) object).getValue();

    setPersistenceProperties();

    final EntityManager entityManager = entityManagerFactory.
        createEntityManager();

    entityManager.getTransaction().begin();
    entityManager.persist(employee);
    entityManager.getTransaction().commit();
    entityManager.close();

    hjid = employee.getHjid();
} ...

return hjid;
}

public void update(String employeeString, Long hjid) {
    final Object object;
    final EmployeeType employee;
    try {
        final Unmarshaller unmarshaller = context.createUnmarshaller();
        InputStream inputStream = new ByteArrayInputStream(employeeString
            .getBytes("UTF-8"));
        object = unmarshaller.unmarshal(inputStream);
        employee = ((JAXBElement<EmployeeType>) object).getValue();

        employee.setHjid(hjid);

        setPersistenceProperties();

        final EntityManager entityManager = entityManagerFactory.
            createEntityManager();

        entityManager.getTransaction().begin();
        entityManager.merge(employee);
        entityManager.getTransaction().commit();
        entityManager.close();
    }
}
```

3. Integration Approach

```
} ...  
}  
  
...  
}
```

Listing 3.9: EmployeeXmlAdapter.java

3.3.3. Conclusion and Deployment

After implementing the persistence properties and the two data access objects, the implementation of the persistence layer is completed. As intended, this component represents a uniform and consistent database access for both YAWL workflows and Liferay portlet applications. It also provides a high level of flexibility. Changes in the business object (e.g., an additional attribute) can be applied by running a new build, and require no changes in the DAOs. Because the data access objects are implemented manually, potentially needed additional methods can be added quickly and easily.

The persistence layer is used by both of the other main components. So, it has to be added as dependency to be available at runtime. The portlet application project, which is discussed later in this chapter, includes the persistence layer automatically during the build process. Like all YAWL components, the Resource Service has been obtained as finished web application and all additional files must be added manually. So, the compiled Java archive file has to be copied to the dependency folder of the Resource Service (e.g., `/opt/lr622yawl3/tomcat-7.0.42/webapps/resourceService/WEB-INF/lib`, cf. figure 3.6).

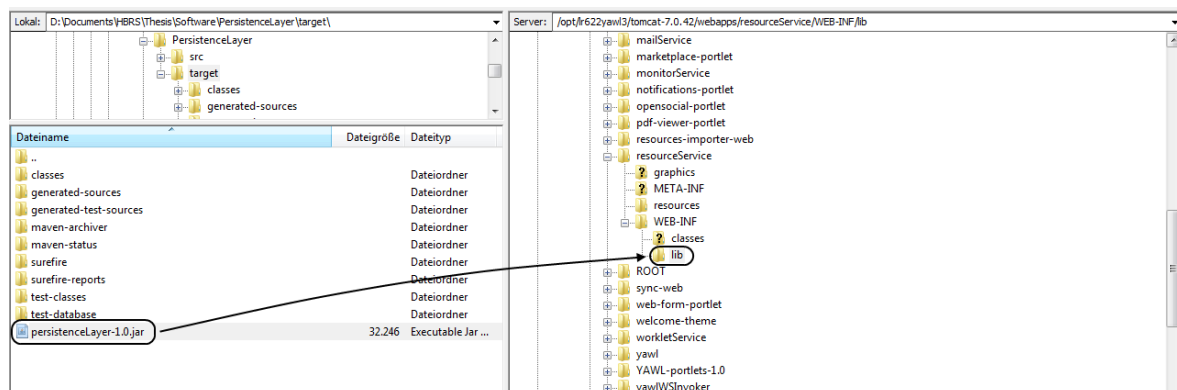


Figure 3.6.: Persistence Layer Deployment to YAWL

3.4. YAWL Extensions

The YAWL extensions represent another component of the draft overall architecture. All classes which are provided inside YAWL packages or are deployed to a YAWL application belong to this component.

3. Integration Approach

3.4.1. Organizational Data Import

By default, YAWL has its own organizational data management and stores users, roles, etc. in its database. In this examination, the organizational data is exclusively managed by the portal and shared to the YAWL WMS. As mentioned in the second chapter of this thesis, YAWL provides a mechanism to import external organizational data by replacing the responsible Java class. Because theoretical aspects like the mapping of one organizational model to another have already been discussed, the pure implementation of the respective classes is shown here.

While all regarded previous examinations [Ros13, Dam14, Mor14] used Java Database Connectivity (JDBC) to query the portal's database directly for organizational data entries, in this examination a more sophisticated and proper way is used. This is possible because YAWL and Liferay are deployed to the same web application server. So, YAWL can access Liferay's service APIs which are available for all applications on the Apache Tomcat which the portal runs on. The relevant services for this part of the integration are *PortalUtil*, *UserLocalServiceUtil* and *RoleLocalServiceUtil*.

3.4.2. Liferay Portal Orgdata Class

The *LiferayPortalOrgdata.java* extends YAWL's abstract *DataSource.class* and replaces the *HibernateImpl.class* which is the default, and manages the organizational data by use of the YAWL database. Because the orgdata has to be shared only one way (i.e., from Liferay to YAWL), all methods regarding creation, persistence, and manipulation of organizational data can be left out. The methods implemented for this examination are shown in listing 3.10.

The basic attention in this class should be paid to the *loadResources()* method. It creates, populates, and returns a *ResourceDataSet* which is the default object for YAWL organizational data. Because capabilities, organization groups, and positions are not taken into account (cf. figure 2.1), the respective hash maps stay empty. The external roles and users are added using the auxiliary methods *loadRoles* and *loadUsers*. By setting *setAllowExternalOrgDataMods(false)* YAWL is notified that it is not allowed to change any data in the resource data set. Because the portal has a comprehensive and well working authentication mechanism, the external user authentication is enabled. This means user names and passwords are forwarded to the external organizational data source (i.e., to the portal) for validation. The overridden method *authenticate(String userid, String password)* does exactly this.

```
public class LiferayPortalOrgdata extends DataSource {  
  
    ...  
  
    @Override  
    public ResourceDataSet loadResources() {  
        ResourceDataSet resourceDataSet = new ResourceDataSet(this);  
        resourceDataSet.setCapabilities(new HashMap<String, Capability>(),  
            this);  
        resourceDataSet.setOrgGroups(new HashMap<String, OrgGroup>(), this);  
    }  
}
```

3. Integration Approach

```
resourceDataSet.setPositions(new HashMap<String, Position>(), this);
resourceDataSet.setRoles(loadRoles(), this);
resourceDataSet.setParticipants(loadParticipants(resourceDataSet),
    this);
resourceDataSet.setAllowExternalOrgDataMods(false);
resourceDataSet.setExternalUserAuthentication(true);
//System.out.println("[LiferayPortalOrgdate] refreshed " + Time.
    getShortTimestamp());
return resourceDataSet;
}

public HashMap<String, Participant> loadParticipants(ResourceDataSet
resourceDataSet) {
    HashMap<String, Participant> hashMap = new HashMap<>();
    List<Participant> participants = liferayResourcesAdapter.
        getParticipants();
    for (Participant participant : participants) {
        List<String> roleIds = liferayResourcesAdapter.
            getRoleIdsByParticipantID(participant.getID());
        for (String roleId : roleIds) {
            Set<Role> roles = resourceDataSet.getRoles();
            for (Role role : roles) {
                if (roleId.equalsIgnoreCase(role.getID())) {
                    participant.addRole(role);
                }
            }
        }
        //System.out.println("ADDED PARTICIPANT ID: " + participant.getID
            ());
        hashMap.put(participant.getUserID(), participant);
    }
    return hashMap;
}

public HashMap<String, Role> loadRoles() {
    HashMap<String, Role> hashMap = new HashMap<>();
    List<Role> roles = liferayResourcesAdapter.getRoles();
    for (Role role : roles) {
        hashMap.put(role.getID(), role);
        //System.out.println("... role: "+role.getName());
    }
    return hashMap;
}

...

@Override
```

3. Integration Approach

```
public boolean authenticate(String userid, String password) throws
    YAuthenticationException {
    return liferayResourcesAdapter.authenticate(userid, password);
}
}
```

Listing 3.10: LiferayPortalOrgdata.java

3.4.3. Liferay Resources Adapter Class

The *LiferayResourcesAdapter.java* (cf. figure 3.11) represents the data access object for the *LiferayPortalOrgdata* class. It uses the above-mentioned Liferay service APIs to fetch the organizational data from the portal. This class also handles the mapping of Liferay's organizational model to that of YAWL and the user authentication. The *authenticate(String screenName, String password)* method is able to validate passwords which are encrypted or unencrypted (i.e., clear text). This is done by either comparing the encrypted password stored in the Liferay user object with the given value or by calling an authentication method which requires the password in clear text.

Special attention should be paid to the detailed mapping of Liferay users to YAWL participants and the assignment of privileges depending on a user's role. Because Liferay has no equivalent of YAWL's user privileges, a non-administrative user gets two privileges, while a user with administrative rights gets all. Of course, this mechanism could be extended with more roles, between which a distinction is made. It is not possible, however, to grant privileges to individual users regardless of their role.

```
public class LiferayResourcesAdapter {

    ...

    public List<Participant> getParticipants() {
        List<Participant> participants = new ArrayList<>();
        //clear database cache to get actual data
        CacheRegistryUtil.clear();
        List<User> liferayUsers;
        try {
            liferayUsers = userLocalService.getCompanyUsers(companyId,
                QueryUtil.ALL_POS, QueryUtil.ALL_POS);
            for (User user : liferayUsers) {
                //map basic values
                Participant participant = new Participant();
                participant.setID(user.getScreenName());
                participant.setUserID(user.getScreenName());
                participant.setFirstName(user.getFirstName());
                participant.setLastName(user.getLastName());
                participant.setPassword(user.getPassword());
                participant.setDescription(user.getJobTitle());
                participant.setNotes(user.getComments());
            }
        } catch (Exception e) {
            // ...
        }
        return participants;
    }
}
```

3. Integration Approach

```
//define privileges for standard user
UserPrivileges privileges = new UserPrivileges();
privileges.setCanReorder(true);
privileges.setCanChooseItemToStart(true);
privileges.setCanChainExecution(false);
privileges.setCanManageCases(false);
privileges.setCanStartConcurrent(false);
privileges.setCanViewOrgGroupItems(false);
privileges.setCanViewTeamItems(false);
participant.setUserPrivileges(privileges);
//check if liferay user is admin, if yes -> more privileges
List<com.liferay.portal.model.Role> liferayRoles = user.
    getRoles();
for (com.liferay.portal.model.Role role : liferayRoles) {
    if (role.getName().equalsIgnoreCase("administrator")) {
        participant.setAdministrator(true);
        participant.getUserPrivileges().setCanChainExecution(
            true);
        participant.getUserPrivileges().setCanManageCases(true)
            ;
        participant.getUserPrivileges().setCanStartConcurrent(
            true);
        participant.getUserPrivileges().setCanViewOrgGroupItems(
            true);
        participant.getUserPrivileges().setCanViewTeamItems(
            true);
    }
}
participants.add(participant);

}

} ...
return participants;
}

public List<Role> getRoles() {
    List<com.liferay.portal.model.Role> liferayRoles;
    List<Role> roles = new ArrayList<>();
    try {
        liferayRoles = RoleLocalServiceUtil.getRoles(companyId);
        for (com.liferay.portal.model.Role liferayRole : liferayRoles) {
            Role role = new Role();
            role.setName(liferayRole.getName());
            role.setID(liferayRole.getName().toLowerCase());
            role.setDescription(liferayRole.getDescription());
            roles.add(role);
        }
    }
```

3. Integration Approach

```
    } ...  
    return roles;  
}  
  
...  
  
public boolean authenticate(String screenName, String password) {  
    boolean returnValue = false;  
    String encPwd;  
    try {  
        encPwd = userLocalService.getUserByScreenName(companyId,  
            screenName).getPassword();  
        if (userLocalService.authenticateByScreenName(companyId,  
            screenName, password, null, null, null) == 1) {  
            returnValue = true;  
        } else if (encPwd.equalsIgnoreCase(password)) {  
            System.out.println("encoded authentication succesful");  
            returnValue = true;  
        }  
    } ...  
    return returnValue;  
}  
}
```

Listing 3.11: LiferayResourcesAdapter.java

3.4.4. Persistence Codelets

The workflow shown in figure 3.4 requires two different persistence codelets. Subnet-task *Initialize* contains an automated task for loading an employee data set to the workflow and the *Finalize* subnet-task is responsible for writing this employee business object back to the database. Figure 3.7 illustrates how the persistence codelets interact with the automated tasks and the persistence layer.

A codelet has to implement YAWL's *AbstractCodelet* interface. The *execute* method is the central point in this class. As the name suggests, this method is called when the Resource Service executes the automated task to which the codelet is assigned to. To ensure a correct execution of the portlet, the input and output have to match input and output of the automated task exactly (cf. figure 3.8).

The Listing 3.12 shows the *execute* method in the *EmployeeReaderCodelet.java*. The codelet receives the *hjid* input variable (i.e., the primary key of the requested data set) as input parameter and calls *EmployeeXmlAdapter* as the corresponding persistence layer class. Using the respective methods of this class, the codelet first blocks the employee and then obtains it as an XML-containing string. This string is converted to an XML-representing object and returned to the automated task. The auxiliary method, which handles the conversion, is not included in the listing but in the provided source code.

3. Integration Approach

The second required persistence codelet works the other way round (cf. listing 3.12). It receives the employee data set as child of an input element and its primary key as input parameter. The child element is converted to an XML-containing string and handed over to the respective method of *EmployeeXmlAdapter* together with its primary key. The *blocked* value is set back to false and an empty element is returned (i.e., there is no value given back to the workflow). This corresponds to the variables of the task the codelet is assigned to (i.e., two input and no output variables are defined).

```
@Override
public Element execute(Element inData, List<YParameter> inParams, List<
    YParameter> outParams) throws CodeletExecutionException {
    setInputs(inData, inParams, outParams);

    Long hjid = Long.valueOf((String) getParameterValue("hjid"));

    Element outputData = new Element("codelet-output");
    EmployeeXmlAdapter employeeAdapter = new EmployeeXmlAdapter();
    //block dataset
    employeeAdapter.setBlocker(hjid, true);
    //read dataset
    String employeeString = employeeAdapter.read(hjid);
    Element employeeElement = stringToElement(employeeString);

    outputData.addContent(new Element("employee"));
    outputData.getChild("employee").addContent(employeeElement.
        cloneContent());

    return outputData;
}
```

Listing 3.12: EmployeeReaderCodelet.execute(...)

```
@Override
public Element execute(Element inData, List<YParameter> inParams, List<
    YParameter> outParams) throws CodeletExecutionException {
    setInputs(inData, inParams, outParams);

    Long hjid = Long.valueOf((String) getParameterValue("hjid"));

    XMLOutputter xmlOutputter = new XMLOutputter();

    Element personalInformationElement = inData.getChild("employee");

    EmployeeXmlAdapter employeeAdapter = new EmployeeXmlAdapter();
    employeeAdapter.update(xmlOutputter.outputString(
        personalInformationElement), hjid);
    employeeAdapter.setBlocker(hjid, false);
}
```

3. Integration Approach

```
return new Element("codelet-output");  
}
```

Listing 3.13: EmployeeUpdaterCodelet.execute(...)

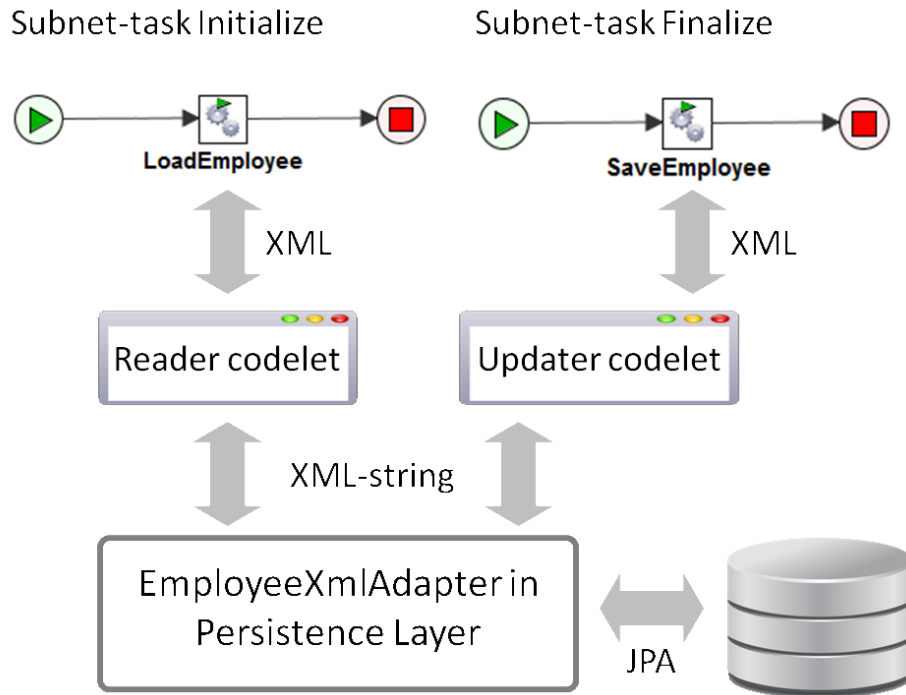


Figure 3.7.: Persistence codelets

3.4.5. Portal Communication Codelet

The sample workflow (cf. figure 3.4) requires an additional codelet, which does not communicate with the persistence layer, but with Liferay Portal. As discussed in the third chapter of this thesis, an automated task (and therefore a codelet) shall generate user name and password for the new employee. Based on these credentials, a new account in the portal has to be added. This allows the new employee to participate in the current workflow by filling out a form by himself.

In contrast to the persistence codelets, this one contains several important methods, which are shown in listing 3.14. When executed, the codelet receives the new employee's hjid (i.e., the primary key), first name, and last name as input parameters. Based on these values, the method *createUserId* creates a user ID consisting of the first letter of the first name, the three first letters of the last name, and the hjid. To give an example, *John Smith* with hjid *1234* would result in the user ID *jsmi1234*. The password is represented by an eight digit random number (method *createPassword*). Because it has to be passed on to the new employee by the HR-Administrator and can be changed by the user after his first login, such a disposable password is considered to be the right solution.

3. Integration Approach

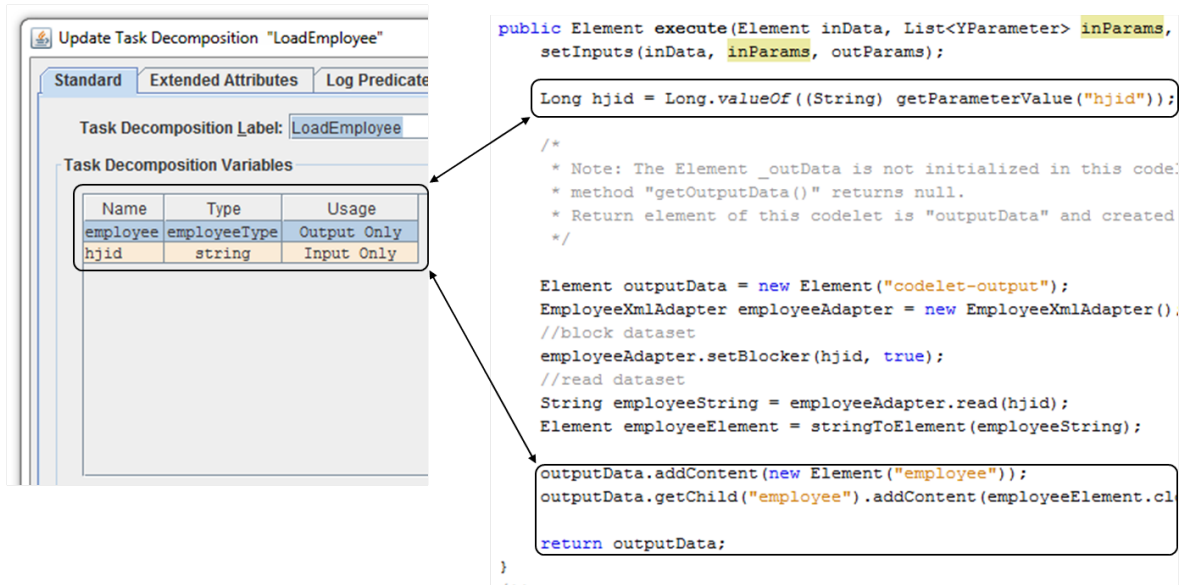


Figure 3.8.: Matching variables in task and codelet

The *createUser* method performs the communication with the portal by calling the respective method in Liferay's service APIs. This method is the original user creation method of Liferay Portal and requires (as many other Liferay methods) a comparatively large amount of parameters. As can be seen in listing 3.14, some of these parameters need to be initialized with or without values (e.g., *groupIds*, *userGroupIds*) while other ones can be derived from the context (e.g., *email address*) or are filled with dummy values (e.g., *facebookId*).

By use of these methods, *execute* performs the generation of user ID and password and the creation of the new account in the portal. If the portal's *createUser* method does not succeed, the homonymous method in the codelet returns a false boolean value, and an error message is returned to the workflow instead of the password. Because the Liferay user creation method is supposed to be robust and deterministic, it should only fail if one or more of the values which have to be unique (e.g., the user id) already exist in the portal. Because the primary key of the employee's data set is involved in the creation, the user ID is absolutely unique in case of a normal workflow termination. But when the workflow breaks off after the execution of this portlet and is started again with the same data set (i.e., with the same name and primary key), the user creation method will not succeed and the error message is returned. In such a case, an administrator has to set a new disposable password in the portal's user management before the new employee can continue the workflow.

```
@Override
public Element execute(Element inData, List<YParameter> inParams, List<
    YParameter> outParams) throws CodeletExecutionException {
    setInputs(inData, inParams, outParams);
    //get incoming parameters
    Long hjid = Long.valueOf((String) getParameterValue("hjid"));
    String firstName = (String) getParameterValue("firstName");
```

3. Integration Approach

```
String lastName = (String) getParameterValue("lastName");
//generate userId
String userId = createUserId(firstName, lastName, hjid);
setParameterValue("userId", userId);
String password = createPassword();
if (createUser(userId, password, firstName, lastName)) {
    setParameterValue("password", password);
} else {
    setParameterValue("password", "already known or error");
}

return getOutputData();
}

private String createUserId(String firstName, String lastName, Long hjid)
{
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.append(firstName.toLowerCase().charAt(0));
    stringBuilder.append(lastName.toLowerCase().subSequence(0, 3));
    stringBuilder.append(hjid);
    return stringBuilder.toString();
}

private String createPassword() {
    return RandomStringUtils.randomNumeric(8);
}

private boolean createUser(String userId, String password, String
    firstName, String lastName) {
    boolean success = false;
    try {
        long[] groupIds = {CompanyLocalServiceUtil.getCompanyById(
            companyId).getGroup().getGroupId()};
        long[] organisationIds = {};
        long[] roleIds = {RoleLocalServiceUtil.getRole(companyId, "User")
            .getRoleId()};
        long[] userGroupIds = {};
        ServiceContext serviceContext = new ServiceContext();

        userLocalService.addUser(userLocalService.getDefaultUserId(
            companyId), companyId, false, password, //long creatorUserId,
            long companyId, boolean autoPassword, String password1
            password, false, userId, userId + "@" +
            CompanyLocalServiceUtil.getCompanyById(companyId).
            getWebId(), //String password2, boolean autoScreenName,
            String screenName, String emailAddress,
            OL, "", LocaleUtil.getDefault(), firstName, "", //long
            facebookId, String openId, Locale locale, String
```

3. Integration Approach

```
        firstName, String middleName,
        lastName, 123, 456, true, 1, 1, //String lastName, int
        prefixId, int suffixId, boolean male, int birthdayMonth
        , int birthdayDay,
        2014, "New Employee", groupIds, organisationIds, roleIds,
        //int birthdayYear, String jobTitle, long[] groupIds,
        long[] organizationIds, long[] roleIds,
        userGroupIds, false, serviceContext); //long[] userGroupIds
        , boolean sendEmail, ServiceContext serviceContext
        success = true;
    } ...
}
```

Listing 3.14: CreateCredentialsCodelet.java

3.4.6. Style Adjustment by CSS

As a matter of fact, the default user interfaces of the YAWL applications can be described as a little dusty and outdated. Of course, the intention of YAWL is to be an open source WMS with high flexibility, expressiveness and functionality, not to provide some good-looking user interfaces. But in this examination, YAWL is integrated into the current version of a portal which provides modern-styled and good-looking user interfaces, and it is expected that one or more artifacts of the Resource Service user interfaces are visible in the portal environment.

At the end of the day, browser-based user interfaces are no more than web pages, and like web pages, the Resource Service JSPs are styled with CSS (Cascading Style Sheets). The corresponding *stylesheet.css* file can be found in the *resources* folder of the Resource Service web application (e.g., */opt/lr622yawl3/tomcat-7.0.42/webapps/resourceService/resources*). The CSS code is not discussed in detail at this point, but is available as part of the provided source code. The images 3.9, 3.10 and 3.10 give an impression of the changes applied to some parts of the YAWL user interface.

The Resource Service has been equipped with an appearance similar to the current Liferay style. The colors of menu bar, menu items, and buttons correspond to the counterparts in the portal. But because the Resource Service user interface is not intended to be visible for non-administrative users, its new style is just a side effect of the adjustment of a much more important component: the work item form. For every task that provides a user interaction, a form is generated by YAWL and displayed as part of a portlet application. Hence, this form page is visible to all users participating in workflows and must fit in as smoothly as possible. Therefore, the YAWL banner is removed, as well as the color gradations between subforms. With white as main color and the blue buttons, the form fits seamlessly into the portal environment (cf. fig. 3.11).

3.4.7. Conclusion and Deployment

The organizational data import, discussed at the beginning of this chapter, represents a clean and proper way to transfer Liferay Portal orgdata to the YAWL WMS. Instead of

3. Integration Approach



Figure 3.9.: YAWL Resource Service default style

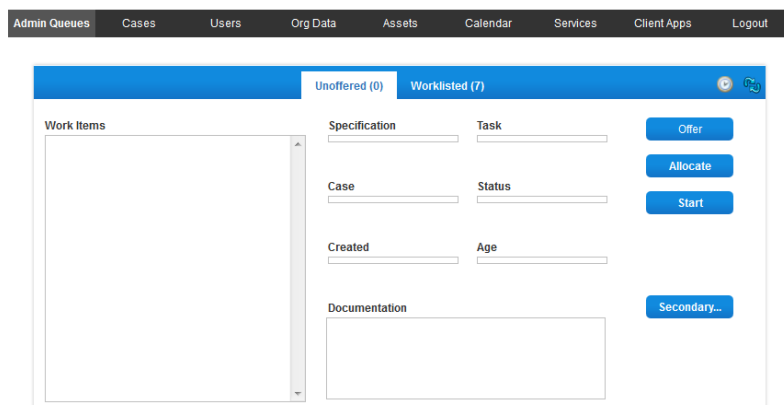



Figure 3.10.: YAWL Resource Service adjusted style

3. Integration Approach



YAWL

Edit Work Item: 14.2

work

Finish: ☐

employee

firstName:

lastName:

dateOfBirth: 25.06.2014

nationality:

gender: <-- Choose (optional) -->

privateContacts

eMail:

phone:

mobile:

officialContacts

phone:

eMail:

blocked: ☐

workflowStatus:

Cancel Save Complete

Edit Work Item: 9.5

work

Finish: ☐

employee

firstName: John

lastName: Smith

dateOfBirth: 27.05.2014

nationality:

gender: <-- Choose (optional) -->

privateContacts

eMail:

phone:

mobile:

officialContacts

phone:

eMail:

blocked: ☐

workflowStatus:

Cancel Save Complete

Figure 3.11.: YAWL form default and adjusted style

3. Integration Approach

Table 3.2.: Changes in Resource Service web.xml

Parameter name	Default value	New value
OrgDataSource	HibernateImpl	LiferayPortalOrgdata
ExternalUserAuthentication	false	true
OrgDataRefreshRate	-1 (disabled)	1 (one minute)

querying the portal's database, the provided portal service APIs are used for this purpose. These are also successfully used for creating a new portal user account in the portal communication codelet. Hence, the portal and the WMS can communicate bidirectionally and exchange data with each other without mutual database access.

The persistence codelets have been thoroughly tested and no limitations could be identified. So, they represent a well working mechanism to access external data sources (e.g., a database) from a running workflow instance. Because the YAWL user interface can be styled by CSS, an integration of visible components into another web application is feasible.

As already mentioned, the YAWL Resource Service is a finished web application, and all additional files must be added manually. This also applies to the YAWL extensions discussed in this section. They have to be copied to folders of the Resource Service which correspond with their packages.

- *LiferayPortalOrgdata.class* and *LiferayResourcesAdapter.class* are copied to */resourceService/WEB-INF/classes/org/yawlfoundation/yawl/resourcing/datastore/orgdata*.
- All codelet class files go to */resourceService/WEB-INF/classes/org/yawlfoundation/yawl/resourcing/codelets*
- *stylesheet.css* replaces the */resourceService/resources/stylesheet.css* file.

To enable the automatic organizational data import, it is necessary to change some settings in the *web.xml* of the Resource Service web application. The default settings shown in listing 2.2 are changed according to table 3.2.

3.5. Implementation of Portlet Applications

As discussed in the third chapter of this thesis, the portlet applications are implemented using the Vaadin web application development framework [Grö13, cf. pp. 363-385]. The complete portlet project is available as part of the provided source code. The four codelets can be divided into two categories (cf. fig. 2.7). One category is represented by the YAWL integration portlets, which interact with the WMS and provide information about the workqueue and assigned work items. They also allow the respective user to process an assigned work item without leaving the portal. The other category includes the business application portlets, which provide access to the data collected or completed by YAWL workflows. In this section, only a small and generic part of the source code is listed. To obtain

3. Integration Approach

details about the detailed implementation of the individual portlets and auxiliary classes, an in-depth look at the source code, which is provided on the attached data carrier and as download, is recommended.

3.5.1. Generic Portlet Body

In principle, all implemented portlets are based on the same structure, which has been elaborated upon during this examination and is shown in listing 3.15. The most important part of a codelet is the *init* method, which is called every time the portlet is accessed in a new user session. This means, for example, that if a user first opens the portlet page anonymously and then logs in staying on the page, the *init* method is called twice. So, this method is the right place for all actions to be taken only once in the same session.

The implemented portlets make use of some addons to augment their abilities. In the current version, Vaadin supports no server push for portlets. This means that the user interface will not update automatically if any data on the server changes (e.g., a new workitem is available). In addition, the YAWL WMS does not support any server-side push functionality, and therefore a push service for the portlets would not make sense even if it were available. The Vaadin refresher addon helps to solve this problem. Per se, it is not more than a polling mechanism, but it offers the possibility to attach listeners. This means, for example, that when a polling is done and the page is reloaded, a method is triggered which gets the current available work items from the WMS. The first section of the listing 3.15 enables this refresh mechanism. A new refresher instance is set to an interval of 5000ms (i.e., five seconds) and a listener object with an overridden refresh method is attached. As long as the refresher is active, this method will be called every 5 seconds.

Vaadin Portlets are individual web pages and therefore independent and separate from each other. So, by default they are not even able to communicate with each other. But in this integration approach, the portlets belonging to the same category have to interact with each other. This means that the two YAWL interaction portlets shall communicate with each other as well as the two business application portlets. The Liferay IPC *inter portlet communication*, which is also a Vaadin addon, bridges that gap. It enables a portlet to send and receive labeled messages. The second section in listing 3.15 gives an example of setting up the IPC. When the portlet is attached to a new IPC instance, it is able to send messages. The added listener enables the portlet to receive all messages labeled with *do the following* and perform the respective action.

A common portlet application is able to switch to different portlet modes (i.e., views). Usually, the main content is provided by the *view mode* which is therefore indispensable. Settings for the current portlet instance should be provided by the *edit mode* and any additional information should be placed in the *help mode*. To support mode changes, a Vaadin portlet has to implement a *PortletListener* interface (see l. 1 in listing 3.15) and must be attached as listener to the portlet session as done in the third section.

As can be seen in the fourth section of listing 3.15, the content for all three portlet modes is defined in the *init* method. Every portlet mode is represented by a single layout to which the respective content is added. The last section in the listing shows how the portlet modes are switched when triggered by a request. It should be mentioned that there is no

3. Integration Approach

additional HTML or JSP definition for a portlet. As discussed in section 2.7.2 on page 20, Vaadin portlets are almost completely implemented in Java and the pages are rendered by Javascript in the client browser.

```
public class GenericPortletBody extends UI implements PortletListener {
    ...

    @Override
    protected void init(VaadinRequest request) {
        //allow this portlet to refresh automatically
        refresher = new Refresher();
        refresher.setRefreshInterval(5000);
        refresher.addListener(new Refresher.RefreshListener() {
            @Override
            public void refresh(Refresher source) {
                //load some data
            }
        });

        //allow this portlet to receive messages from other portlets
        final LiferayIPC ipc = new LiferayIPC();
        ipc.extend(this);
        ipc.addLiferayIPCEventListener("do the following", new
            LiferayIPCEventListener() {
                @Override
                public void eventReceived(LiferayIPCEvent event) {
                    String url = event.getData();
                    //do something
                }
            });

        // Check that we are running as a portlet.
        if (VaadinSession.getCurrent() instanceof VaadinPortletSession) {
            VaadinPortletSession portletsession =
                (VaadinPortletSession) VaadinSession.getCurrent();
            // Add a custom listener to handle action and
            // render requests.
            portletsession.addPortletListener(this);
        }

        ...

        //view content
        viewContent = new VerticalLayout();
        viewContent.addComponent(new Label("view content"));

        //editContent
        editContent = new VerticalLayout();
```

3. Integration Approach

```
editContent.addComponent(new Label("edit content"));

//helpContent
helpContent = new VerticalLayout();
helpContent.addComponent(new Label("help content"));

setContent(viewContent);

}

...

// Switch the view according to the portlet mode
@Override
public void handleResourceRequest(ResourceRequest request,
    ResourceResponse response, UI uI) {
    if (request.getPortletMode() == PortletMode.EDIT) {
        this.setContent(editContent);
    } else if (request.getPortletMode() == PortletMode.VIEW) {
        this.setContent(viewContent);
    } else if (request.getPortletMode() == PortletMode.HELP) {
        this.setContent(helpContent);
    }
}
}
```

Listing 3.15: Generic portlet body

3.5.2. Workqueue Portlet

The workqueue portlet is by far the most complex of the four portlets implemented for this examination. It is completely generic and adjustable, and can interact with any YAWL WMS that is accessible via local or external network. It supports YAWL as standalone WMS and integrated to Liferay Portal. For accessing an external WMS, the user only requires the network address, his user ID, and his password. For more details, a look at the respective source code is recommended. Figure 3.12 shows the workqueue portlet in view mode.

As can be seen on the upper right notification, a portal user is logged in. By default, the workqueue portlet lists all available work items which are assigned to this user or his role. The work items are sorted in four different tables depending on their status. Every work item has a drop down menu which offers actions according to the current status. For example, an offered work item can be accepted or started that changes its status in the workqueue. An advantage of this portlet compared to the original YAWL workqueue application is the possibility to start working on any work item regardless of their statuses. If there is a table without work items, a message is displayed and a refresh is offered instead of an actions menu.

When *work on task* is selected, the YAWL Resource Service provides a generated form

3. Integration Approach

which is accessible via a specific URL. To access this form, the workqueue portlet offers three alternatives which can be selected in the edit mode. By default, an IPC message with the URL is sent to the workitem portlet which opens the form inside the portal. Alternatively, the form can be opened in a separate browser window or the form URL can be delegated to another device via QR code or link (cf. fig. 3.13).

3.5.3. Workitem Portlet

As mentioned above, the workitem portlet is responsible for displaying a selected workitem inside the portal. When it receives an IPC message labeled with *openUrl*, the provided URL is opened in an iframe. As can be seen in figure 3.14 the form fits seamlessly into the portlet. This is an outcome of the applied CSS styling which has been discussed in section 3.4.6 on page 52. Without being restyled, the form would appear like the left part of figure 3.11, which obviously would not fit in without significant optical deviations.

The three buttons next to the form belong to the YAWL page inside the iframe and are required to control the processing of the work item. When the *complete* button is pushed, YAWL checks the work item back into the engine and asks the user to close the window. Instead of removing the iframe from the portlet, a HTML placeholder page is loaded by clicking on *close workitem* (which is a portlet component outside the iframe). This placeholder page is shown in figure 3.15.

3.5.4. Data Overview Portlet

The data overview portlet shall provide an overview of the business data sets (i.e., in this case the employees) which have been initially created or completed by YAWL workflows. Figure 3.16 shows the employees who are present in the personnel database. Like the workitems in the workqueue portlet, every data set has its own drop down actions menu. The actions offered by this menu depend on the *workflowStatus* and on the *blocked* value in the employee object (cf. sec. 3.2). If an employee has been created but not yet checked-in, it is possible to view details, to start the check-in process and to delete the data set. To start a check-out process is only possible when the check-in has already been done. If the *blocked* flag is set, the actions menu only allows to watch the employee's details.

When *new employee* is selected in the main actions menu, a pop-up window appears and asks for the basic values of the new employee. As discussed in section 3.2, these values are mandatory to provide distinguishability. When the *submit* button is clicked, a new employee object is created, populated with the basic data and written to the personnel database. If selected by activating the checkbox, a new check-in workflow for this employee is started immediately when the form is submitted. Otherwise, the workflow can be started via the actions menu next to the new dataset which should be displayed right after the next refresh (or within less than five seconds if auto refresh is enabled). When *details* is selected in the data set menu, an IPC message labeled with *fetchHjid* and containing the primary key of the respective employee is sent to the data details portlet.

3. Integration Approach

YAWL Workqueue Portlet

You are logged in as christian

YAWL WMS Workqueue

Started workflow tasks

Workitem	Case Details	Actions
9.5:work	Claas Clever	Actions

Allocated workflow tasks

Workitem	Case Details	Actions
7:work	hein bloed	Actions

Offered workflow tasks

Workitem	Case Details	Actions
12:work	Donald Duck	Actions
15:CheckBasicData	Klaus Klausingen	Work on Task
11:work	Hans Wurst	Accept
10:work	Daniel Duesentrieb	Accept and Start
8:work	Simon Simonis	Actions

Suspended workflow tasks

Workitem	Case Details	Actions
none	no suspended workitems available yet	Refresh

[Refresh Workqueue](#)

Figure 3.12.: Workqueue portlet in view mode

3. Integration Approach

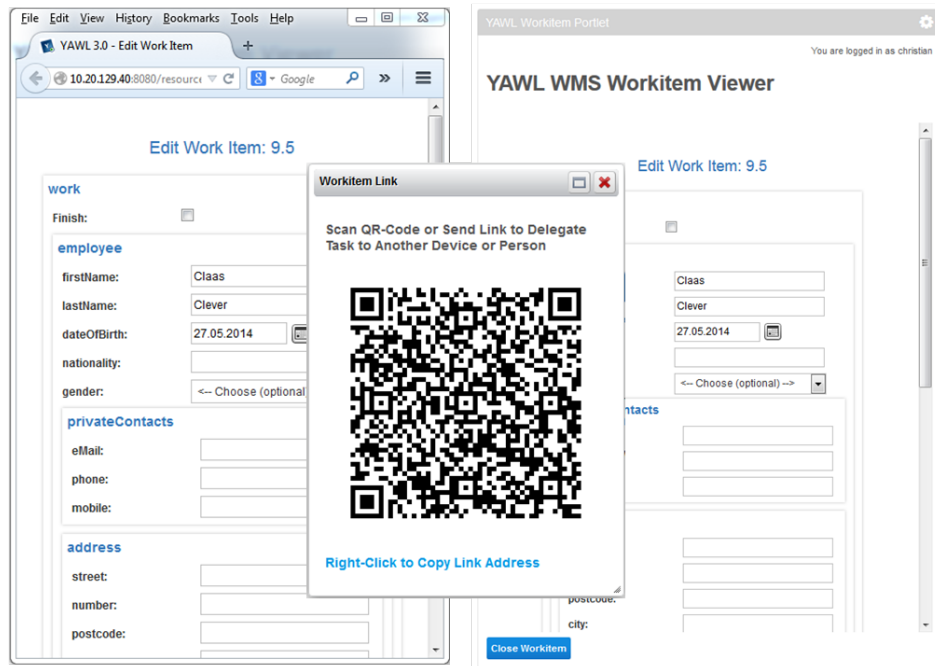


Figure 3.13.: Three alternatives to work on a task

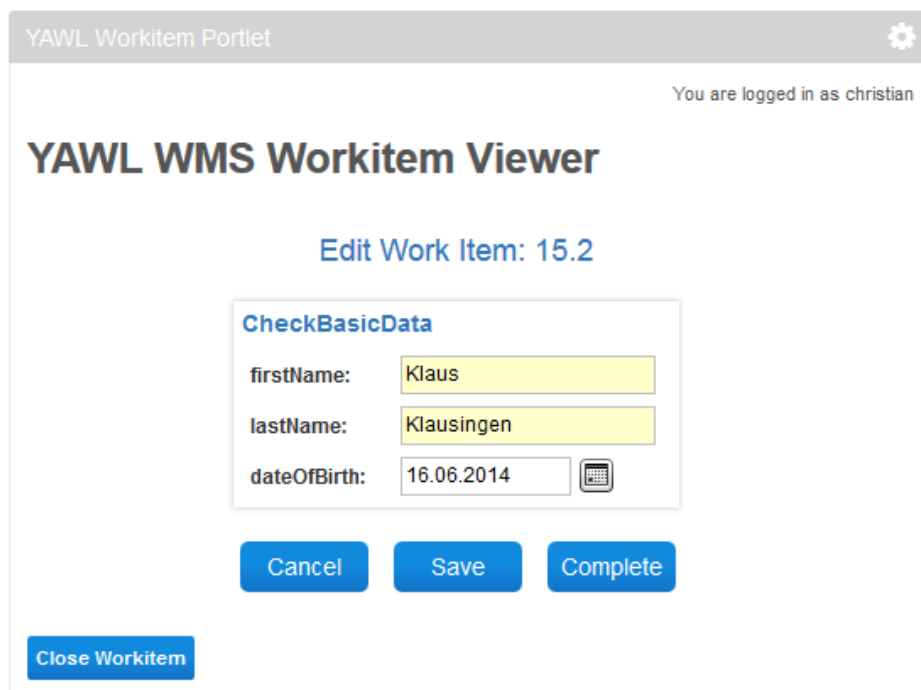


Figure 3.14.: Workitem portlet displaying YAWL form

3. Integration Approach

YAWL Workqueue Portlet

You are logged in as christian

YAWL WMS Workqueue

Started workflow tasks

Workitem	Case Details	Actions
15.2:CheckBasicData	Klaus Klausingen	Actions
9.5:work	Claas Clever	Actions

Allocated workflow tasks

Workitem	Case Details	Actions
7:work	hein bloed	Actions

Offered workflow tasks

Workitem	Case Details	Actions
12:work	Donald Duck	Actions
11:work	Hans Wurst	Actions
10:work	Daniel Duesentrieb	Actions
8:work	Simon Simonis	Actions

Suspended workflow tasks


Workitem	Case Details	Actions
none	no suspended workitems available yet	Refresh

[Refresh Workqueue](#)

YAWL Workitem Portlet

You are logged in as christian

YAWL WMS Workitem Viewer



The YAWL workflow management system will provide Your task here when You select it from the workqueue.

After finishing a task close the workitem via the button below.

[Close Workitem](#)

Figure 3.15.: Workqueue portlet and workitem portlet

Data Overview Portlet

[Actions](#)

Search by Name :

Employees Overview

First Name	Last Name	Date of Birth	Actions
Maja	Majewski	2004-06-15	Actions
Anton	Antonelli	2014-05-20	Details
Willi	Wilhelmy	2004-06-15	Check-In
Klaus	Klausingen	2014-06-16	Delete

[Refresh Data](#)

Figure 3.16.: Data overview portlet with offered actions

3. Integration Approach

The screenshot shows a web application interface titled "Data Overview Portlet". On the left, there is a table with two columns: "First Name" and "Last Name". The table contains five rows of employee data. Below the table is a "Refresh Data" button. To the right of the table is a modal window titled "Insert new Employee's Basic Data". The modal contains three input fields: "First Name" (with the value "John"), "Last Name" (with the value "Smith"), and "Date of Birth" (with the value "24 Jun 2014"). There is a checkbox labeled "Start Check-In Immediately" and a "Submit" button.

First Name	Last Name
Maja	Majewski
Anton	Antonelli
Willi	Wilhelmy
Klaus	Klausingen

Figure 3.17.: Create a new employee data set

3.5.5. Data Details Portlet

Like the workitem portlet, the data details portlet does not do much more than displaying selected data. It receives a command from the data overview portlet to load a certain data set from the database. To display the obtained object, a form is implemented which corresponds to the object (i.e., the employee). This form is implemented manually, which might be expensive when a business object is large or has a high level of complexity. But a manual implementation provides more flexibility regarding labels and layouts. Figure 3.18 shows the upper section with a part of the form and the lower section with the buttons.

When the portlet receives an IPC message from the data overview portlet, which is labeled *fetchHjid*, it loads the object with the given primary key from the personnel database and populates the form with the contained data. The *refresh* button allows to reload the object and is present in any case. The *save data* button is only displayed if the loaded object has a disabled *blocked flag*. As discussed in section 3.2, this shall prevent that the object is changed while a process in the WMS is working on it.

3.5.6. Auxiliary Classes

Apart from the portlet classes, the portlet project contains several additional classes which are indispensable for the application. For example, the adaptive actions dropdown menu for items displayed in the tables is not available as a standard Vaadin component. But because a Vaadin application is implemented in Java, all Vaadin user interface components are Java classes which can be extended the usual way. This is a further crucial advance compared to other web application frameworks. So, the adaptive actions menus, which are *WorkQueue-WorkItemMenu* and *DataDetailsActionsMenu*, are created by extending Vaadin's *MenuBar*

3. Integration Approach


Data Details Portlet

Personal Data

First Name :


Last Name :

Date of Birth :



Nationality

Gender :



Private Contact Details

Private eMail :

Private Phone :

Private Mobile :

Address

Street :

Number :

Postcode :

City :

Country :

Figure 3.18.: Upper and lower Section of data details portlet

3. Integration Approach

class and adding the respective constructors, methods, and variables. All other classes representing specialized user interface components are also created this way.

YAWL Workqueue Adapter

As mentioned in section 3.3.3, the persistence layer's main component is added as dependency to the portlet project, and therefore is included automatically during the build process. The database access in the business application portlets is done by use of the respective data access object (i.e., the *EmployeeAdapter*) and no additional classes are needed. In contrast to the database access, the interaction with the YAWL requires an additional class to bridge the gap between portlets and the WMS. For this purpose, the YAWL developers provide the *WorkQueueGatewayClientAdapter* [Ada13] which gives an external application comprehensive access to the workqueue and the case management.

The *YawlWorkqueueAdapter* in the portlet project extends this class and adds some methods which are useful for the portlet implementation. These are shown in listing 3.16. The *YawlWorkqueueAdapter* immediately connects to the Resource Service corresponding to the parameters of the constructor invocation. These parameters are provided by the portlet, which obtains them from a properties file. When the YAWL connections are changed in the portlet settings, a new adapter object is initialized. Because the workqueue portlet has to open a workitem form provided by the Resource Service (cf. sec. 3.5.2), it needs the respective URL of that form. Method *getWorkItemFormUrl* generates this URL from workitem ID, user ID, and password.

The data overview portlet shall be able to start a new check-in workflow (also denoted as case), and the method *launchCase* takes care of this action. It receives a unique identifier (i.e., the name) of the workflow specification and a string which contains input variables for the respective workflow. In case of the check-in process, the primary key of the respective employee must be provided. The method obtains a list of all workflow specifications uploaded to the engine and searches for the specification with the given identifier. If the respective workflow specification can be found, a new workflow instance is started.

The workqueue portlet has to show case details (e.g., the employee's name) in the workitem tables (cf. fig. 3.12). To achieve this, a little detour has to be taken. A workitem is not aware of any case data, but it contains the ID of the case (i.e., the workflow instance) it is assigned to. The method *getCaseDetails* obtains the case data through this ID, filters out all parameters which are defined in the given string array, and returns their values as whitespace-separated string. Like the other ones, this method is absolutely generic and can be applied in any other business context.

```
public class YawlWorkqueueAdapter extends WorkQueueGatewayClientAdapter {  
  
    private String handle = "";  
    private String externalEngineUrl;  
  
    public YawlWorkqueueAdapter(String internalEngineUrl, String  
        externalEngineUrl, String adminUserName, String adminPassword) {  
        super(internalEngineUrl + "/resourceService/workqueuegateway");  
    }  
}
```

3. Integration Approach

```
this.handle = connect(adminUserName, adminPassword);
this.externalEngineUrl = externalEngineUrl;
}

...

public String getWorkItemFormUrl(WorkItemRecord workItem, String userName
, String password) {
    StringBuilder url = new StringBuilder(externalEngineUrl + "/"
        resourceService/faces/rssFormViewer.jsp");
    url.append("?itemid=")
        .append(ServletUtils.urlEncode(workItem.getID()))
        .append("&userid=")
        .append(ServletUtils.urlEncode(userName))
        .append("&password=")
        .append(ServletUtils.urlEncode(password));
    return url.toString();
}

public String launchCase(String uri, String caseData) {
    String returnValue = "failure";
    try {
        Set<SpecificationData> specs = getSpecList(handle);
        for (SpecificationData spec : specs) {
            if (StringUtils.containsIgnoreCase(spec.getSpecURI(), uri)) {
                String rootNetId = spec.getRootNetID();
                returnValue = launchCase(spec.getID(), "<" + rootNetId + "
                    >" + caseData + "</" + rootNetId + ">", handle);
            }
        }
    } ...
    return returnValue;
}

...

public String getCaseDetails(String rootCaseId, String[] details) {
    StringBuilder returnValue = new StringBuilder();
    Element caseData;
    boolean found = false;
    try {
        caseData = new SAXBuilder().build(new StringReader(getCaseData(
            rootCaseId, handle))).getRootElement();
        List<Element> children = caseData.getChildren();
        for (String detail : details) {
            for (Element element : caseData.getDescendants(Filters.element
                ())) {
                if (StringUtils.containsIgnoreCase(element.getName(),
```

3. Integration Approach

```
        detail)) {
            returnValue.append(element.getText());
            returnValue.append(" ");
            found = true;
        }
    }
} ...

if (found == false) {
    returnValue.append("Specified Details not found in Case");
}
return returnValue.toString();
}

public String getHandle() {
    return this.handle;
}
}
```

Listing 3.16: YawlWorkqueueAdapter.java

3.5.7. Deployment

XML Descriptors

For a portlet application, some essential configuration files are required. The three most important ones are discussed here. The first two are Liferay-specific, while the third is portal-specific (i.e., also applicable in other portal systems). The *liferay-display.xml* (cf. lst. 3.17) is comparatively simple and describes the category under which the portlets shall appear in the dialog for adding portlets to page [SK12, cf. p. 47].

The *liferay-portlet.xml* describes optional Liferay-specific enhancements for Java portlets which are installed on a Liferay Portal server. For example, it can be set whether a portlet is instanceable, which means more than one instance can be placed on one page [SK12, cf. p. 47]. As can be seen in the listing 3.18, the workqueue portlet is not instanceable and an additional CSS file has to be loaded to the page together with the portlet. Although Vaadin portlets use AJAX (Asynchronous JavaScript and XML), this setting must be disabled at this point [Grö13, cf. p. 369].

The *portlet.xml* contains the portlet definition which includes the portlet name, a servlet mapping, the view modes supported by the portlet, and other configuration [Grö13, cf. p. 368]. Listing 3.19 shows this definition for the workqueue portlet. The *UI* parameter refers to the portlet class including the package name. *Widgetset* is a specific parameter for Vaadin portlets and the view modes are supported as implemented in section 3.5.1.

```
<?xml version="1.0"?>
```

3. Integration Approach

```
<!DOCTYPE display PUBLIC "-//Liferay//DTD Display 6.0.0//EN" "http://www.liferay.com/dtd/liferay-display_6_0_0.dtd">

<display>
  <category name="YAWL WMS">
    <portlet id="WorkqueuePortlet" />
    <portlet id="WorkitemPortlet" />
    <portlet id="RestPortlet"/>
    <portlet id="DataOverviewPortlet"/>
    <portlet id="DataDetailsPortlet"/>
  </category>
</display>
```

Listing 3.17: liferay-display.xml

```
<?xml version="1.0"?>
<!DOCTYPE liferay-portlet-app PUBLIC "-//Liferay//DTD Portlet Application 6.0.0//EN" "http://www.liferay.com/dtd/liferay-portlet-app_6_0_0.dtd">

<liferay-portlet-app>
  <portlet>
    <portlet-name>WorkqueuePortlet</portlet-name>
    <instanceable>>false</instanceable>
    <ajaxable>>false</ajaxable>
    <header-portlet-css>/css/styles.css</header-portlet-css>
  </portlet>

  ... same for all other portlets in project ...

</liferay-portlet-app>
```

Listing 3.18: liferay-portlet.xml

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

<portlet-app
  xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">

  <portlet>
    <description>Portlet for managing the YAWL workqueue</description>
    <portlet-name>WorkqueuePortlet</portlet-name>
    <display-name>YAWL Workqueue Portlet</display-name>
    <portlet-class>com.vaadin.server.VaadinPortlet</portlet-class>
    <init-param>
      <name>UI</name>
```

3. Integration Approach

```
<value>org.freihoff.yawl.portlets.WorkqueuePortlet</value>
</init-param>
<init-param>
  <name>widgetset</name>
  <value>com.vaadin.portal.gwt.PortalDefaultWidgetSet</value>
</init-param>
<supports>
  <mime-type>text/html</mime-type>
  <portlet-mode>view</portlet-mode>
  <portlet-mode>edit</portlet-mode>
  <portlet-mode>help</portlet-mode>
</supports>
<portlet-info>
  <title>YAWL Workqueue Portlet</title>
  <short-title>Portlet Short Title</short-title>
  <keywords>put keywords here</keywords>
</portlet-info>
</portlet>

... same for all other portlets in project ...

</portlet-app>
```

Listing 3.19: portlet.xml

Vaadin Addons

As discussed in section 3.5.1 on page 56, the implemented portlets are augmented by some addons. To get them running in the portal, the respective libraries must be copied to the portal's library folder. The following addons have already been downloaded from the Vaadin addons page for the implementation and will be provided as part of the data attached to this thesis.

- *vaadin-ipc-for-liferay-2.0.0.jar* for the inter portlet communication
- *refresher-1.2.3.7.jar* to enable automatic data and page refresh
- *qrcode-2.0.1.jar* for delegating workitems to other devices by QR code

These JAR files are copied to Liferay's dependency folder (e.g., */opt/lr622yawl3/tomcat-7.0.42/webapps/ROOT/WEB-INF/lib*).

Vaadin Control Panel

The Vaadin web applications make use of compiled widgets which have to be available at runtime. In case of a portlet application, the Vaadin widgetset is provided by the portal and needs to be compiled when components change. Because the Vaadin version used for this implementation differs from the one which is provided by the portal by default, the respective libraries have to be updated and the widgetset has to be recompiled. All addons

3. Integration Approach



Figure 3.19.: Vaadin control panel

have to be included in such a procedure.

Version updates and recompiling are done by use of the Vaadin control panel which is shown in figure 3.19. This application needs to be deployed to the portal, and then is available on the administrative configuration page. Here the *liferay-vaadin-plugin.war* is copied to */opt/lr622yawl3/deploy* and can be accessed right after the deployment has finished. Because the portlet project uses Vaadin version 7.1.8, an upgrade to this version is initiated. All available addons are selected and the widgetset compilation is started.

After the compilation is done, the portlet project is deployed to Liferay portal by copying the application WAR file, which is an outcome of the build process, to the above-mentioned deploy folder. Then the individual portlets can be added to the respective portal pages.

3.6. Acute Issues During the Integration

3.6.1. Liferay Hibernate Cache

The *LiferayResourcesAdapter.java* discussed in section 3.4.3 uses a method of Liferay's *UserLocalServiceUtil* class to obtain all current users of the portal. When this method is called again after adding or removing a user, it returns a list with the old content. A correct list is only returned when the server is restarted. This issue could not be reproduced with other applications (e.g., a codelet calling the same method repeatedly always obtains an updated list). Finally, the issue could be solved by clearing Liferay's hibernate cache before calling the method again [Sam13, cf.]. Other developers facing this issue could not be found. Therefore, it is assumed that it represents a rather rare phenomenon which does not require a bug reporting to the Liferay developers.

3. Integration Approach

Table 3.3.: Reported and solved relevant YAWL bugs

Issue	Description
#491 [FA13]	Editor 3.0 Analyse Specification Window does not terminate/close
#504 [FA14a]	Leaving optional textfield empty leads task into nirvana with error message
#512 [FA14b]	Some SpecificationData Getters allways return null
#517 [FA14c]	User privileges are not taken by participant when using external resources data source
#519 [FA14d]	YAWL looses workflow after database foreign key violation
#523 [FA14e]	Editor 3.0 allows no explicite mappings from primitive to complex datatype (worked in 2.3.5)
#524 [FA14f]	Editor 3.0 creates unwanted out-mapping for input only variable

3.6.2. YAWL Issues

As mentioned in section 1.4, YAWL is a system in ongoing development, and with every new extension or improvement new bugs and limitations find their way into this WMS. During the course of this examination, several issues concerning YAWL components occurred and had to be solved.

The YAWL developers are very quick with fixing bugs after they have been reported. But if the continuation of an implementation depends on a certain software component, a discovered bug always means a high time expense. At first, it has to be ensured that the respective component is responsible for the problem (i.e., bugs in the own code must be excluded by testing and code review). Then, the bug has to be localized as precisely as possible and reported to the developers. A provided bugfix has to be tested thoroughly and reworked, if appropriate. All in all, it can be said that the bugs found during this implementation have taken several days of time.

Table 3.3 provides an overview of the YAWL bugs reported during this examination and the preceding research. The fact that these bugs have been found in certain functions which are essential for this integration approach implies that these components have been seriously applied in their current versions for the first time. Otherwise, the respective bugs would have already been reported and solved. Hence, it can be said that this integration approach and the associated software development represent an actual contribution to the further development of YAWL.

3.6.3. Conclusion

The expected overall architecture of the integration approach has been successfully realized by use of the components discussed in chapter 2. During the implementation, additional details of the architecture could be determined. This leads to the final overall architecture shown in figure 3.20. As well as in figure 2.7, machine and web application server are left out in this depiction. During the implementation of the integration components, no significant limitations were determined, and all occurring issues concerning software development could be solved.

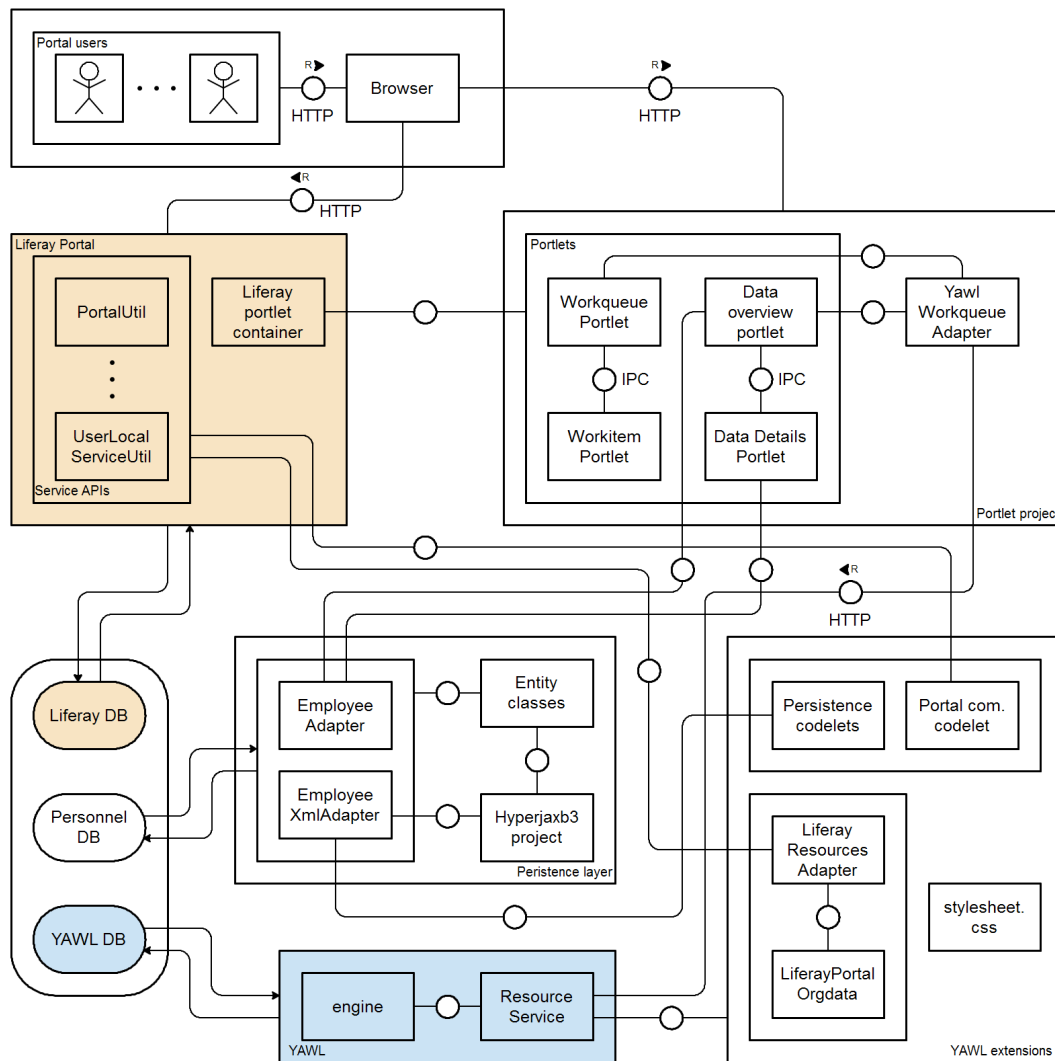


Figure 3.20.: Final overall architecture

4. Results and Evaluation

4.1. Used Technologies

The installation of Liferay Portal and YAWL on one Apache Tomcat web application server proved to be quite feasible, and opened the door to additional advances like the accessibility of portal service APIs for the YAWL extensions. In contrast to previous examinations [Dam14, Mor14], no additional web application server is required and the respective redundancy is prevented. Because all relevant communication between portal and WMS does not leave the web server, no security issues have to be considered. YAWL consists of comparatively lightweight web applications which run beside the Liferay Portal components without any interference or performance issues. Hence, the operation of YAWL and Liferay Portal on one server can be fully recommended.

Apart from the issue discussed in section 3.6.1, no problems with Liferay Portal occurred during the entire examination. This portal proved to be matured and well-working, and therefore provided a reliable basis for the integration of YAWL. The current version offers a wide range of abilities (e.g., responsive design) to operate an advanced and sophisticated business portal environment. Due to the possibilities and capabilities, Liferay's high consumption of resources should be acceptable.

For the author of this thesis, the definition of data types by the use of XML schema definition turned out to be more versatile than expected. A new business object can be created from scratch within a short time and without considering technical aspects (e.g., how to represent a collection of subtypes in a Java class). The XML schema definition language has a comparatively small vocabulary and is therefore easy to learn and to apply. The XSD-defined data type found use in two software components of this integration approach. An additional object or data model was not necessary.

As mentioned in section 2.5, the author of this elaboration developed a first persistence layer for YAWL workflows by the use of Hyperjaxb3 during a previous project. In the course of this examination, the positive expectations regarding this framework were confirmed. The code generation based on the given XSD-defined business object, the use of the generated entity classes, and the conversion from Java object to XML and vice versa worked perfectly and without any problems. Therefore the Hyperjaxb3 project, which was developed by Aleksei Valikov, is an absolutely recommendable software component.

The Vaadin framework kept its promise to be a web application development framework for developing high-quality user interfaces with almost pure Java. An extensive library of user interface components is provided, as well as a large amount of addons for every conceivable purpose. If a very special component is needed, it can be easily created by extending an existing one. Although Liferay supports Vaadin from scratch, a plugin has to be deployed to the portal and some preparatory actions have to be taken to get Vaadin portlets running. However, this effort is kept within limits and is absolutely worth it. For developers who prefer

Java, the Vaadin web application development framework can be recommended.

4.2. Interoperability of YAWL

Apart from the fact that this WMS is open source and can therefore be extended or changed as required, YAWL provides several services and interfaces. If these can be accessed and effectively used by external applications, an existing interoperability can be attested. In section 1.4 some questions, which this examination shall answer, have been asked.

Can users of the portal access and process their assigned work items out of a running workflow inside the portal environment?

The workqueue portlet is able to access the YAWL workqueue and to show all available work items which are assigned to the respective portal user. The statuses of assigned work items can be changed and the work items can be opened to process them. For this purpose, the user can choose from three alternatives (delegate, separate browser window, workitem portlet). The portlet application, as well as the interface classes (i.e., YAWL's *WorkQueueGatewayClientAdapter* and the extending subclass *YawlWorkqueueAdapter*) have been thoroughly tested and operate reliably.

Can workflows deployed to the YAWL WMS be triggered by a portal application?

As discussed in section 3.5.6, the implemented *YawlWorkqueueAdapter* provides a convenient method to start workflows which are uploaded to the engine. Only the identifier (i.e., the unique name) and XML-represented input variables are required. This method is used by the data overview portlet, where the check-in process for a new employee can be started. A specific *workflowStatus* attribute in the business object ensures that only applicable workflows can be started. Because this method is absolutely generic, any other workflow can be triggered by a portlet application.

Can a YAWL workflow and a portal application share the same data on the same database without causing inconsistency?

This integration approach uses a uniform persistence layer which can be used for both, XML and Java objects. The business data is persisted in a separate database instead of storing it as portal content in Liferay's database. So, the data can be accessed from anywhere at any time without limitations. Data can be created by a portlet application (e.g., the data overview portlet), completed by a YAWL workflow and modified by the portlet application again. A special *blocked* flag in the business object prevents that a data set loaded to a running workflow instance can be modified by a portlet application. Hence, data consistency is ensured.

All questions asked at the beginning of this examination can be answered with yes. So, YAWL provides a sufficient interoperability to be integrated into Liferay Portal. In addition, the above-mentioned *WorkQueueGatewayClientAdapter* interface class of YAWL provides much more than the abilities applied in this integration approach. It can be assumed that

4. Results and Evaluation

almost every functionality of the Resource Service, including its interfaces to the YAWL engine, can be used by the provided interfaces.

4.3. Productive Usability of YAWL

As mentioned several times, YAWL is a software in ongoing development. With every new version or improvement, new bugs find their way into the software. Some of them (e.g., [FA14d]) are so significant that the WMS becomes unusable. Fortunately, reported bugs are fixed quickly and the YAWL WMS is brought to a stable version regularly. In addition, most bugs concern the development but not the runtime. In the opinion of the author of this thesis, the YAWL WMS continues working absolutely stable once it is got up and running in a proper configuration.

It should be noticed that YAWL can only be recommended for productive use in a company or organization where someone with advanced knowledge about this WMS is available, but YAWL is open source and therefore free of charge. So, the associated savings should justify the corresponding expenditure for qualified personnel or external consultation.

4.4. Limitations

As discussed in the sections 2.3.1 and 3.4.3, Liferay's organizational model has no equivalent for YAWL's user privileges. Therefore, the privileges are mapped depending on a user's role, and it is not possible to grant privileges to individual users. This limitation is not a problem in this examination, but it may be in another business or organizational context.

The workitem portlet uses an iframe to display workitem forms generated by the Resource Service. This solution works, but might be not the most elegant way. The page in the iframe and the surrounding application are not able to communicate with each other. So, workarounds like the *close workitem* button and the placeholder page are necessary.

4.5. Conclusion

In the course of this examination, a complete and well-working sample application has been developed. It includes the integration of YAWL into a business portal environment by the use of different technologies. The sample application is based on a single web application server where Liferay Portal and YAWL are running side by side. Each of these systems consists of several web applications and uses an individual database exclusively.

Instead of managing its own organizational data, the YAWL WMS obtains users and roles directly from Liferay. This is done by using the provided service API's, which are also used by YAWL workflow instances to add new users to the portal. This means, the two systems are able to exchange data without touching each others databases.

Portlet applications running in the portal can access the WMS for displaying or manipulating the workqueue. Workitems can be opened and processed by the respective user in the portal. New instances of workflows uploaded to the engine can be started by a portlet

4. Results and Evaluation

application. Workflow instances are able to load data from a database and to write it back after manipulating it. This data can also be accessed by portlet applications while data consistency is ensured. The implemented portlets provide a high level of usability and convenience. The workqueue portlet is absolutely generic and can be used to access any other YAWL WMS for every user who is a valid participant there.

The sample application contains no significant limitations and is implemented in a proper and reproducible way. It allows to be reconstructed and to derive usable models for different application contexts.

4.6. Recommendations

Concerning the interaction of YAWL and Liferay Portal, additional research might be recommended. Surely solutions for the limitations determined in this examination exist and should be elaborated. The missing possibility to grant privileges to individual users regardless of their role could be solved by extending the Liferay user class, or by storing privileges as an XML representation in one of the unused attributes.

The workitem forms generated by the Resource Service could be replaced by self-generated forms which can be integrated into the portlet user interface. Therefore, the development of a sophisticated form generator would be required. The outcome of such a development should be independent from the framework used for portlet development.

Both YAWL and Liferay use XML for data definition and carriage. Instead of storing data as Liferay content to the portal database, the Liferay service API's could be used to publish workflow collected data. This way, the cache issue [Dam14, cf. p. 79] could be solved and a persistence layer would no longer be needed. Such an approach would represent an alternative to the separate business database used in this examination. It would require a proper mapping between the XML definitions of both systems.

Bibliography

- [Ada13] M. Adams. WorkQueueGatewayClientAdapter (YAWL Version 2.3), August 2013. Last accessed: 2014-06-28. URL: <http://www.yawlfoundation.org/javadoc/yawl/org/yawlfoundation/yawl/resourcing/rsInterface/WorkQueueGatewayClientAdapter.html>.
- [Boe13] J. Boeder. Making architecture understandable - block diagrams for communication. 2013.
- [Coa99] Workflow Management Coalition. *Workflow Management Coalition Terminology & Glossary*. Workflow Management Coalition, 1999. URL: http://www.wfmc.org/docs/TC-1011_term_glossary_v3.pdf.
- [Dam14] M. Dames. Integration des YAWL-Workflowsystems in die Enterprise-Web-Plattform Liferay durch die Generierung von Benutzeroberflächen und Persistenzschichten. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences, 2014.
- [FA13] C. Freihoff and M. Adams. YAWL Bug Report Discussion: Issue 491 - Editor 3.0 Analyse Specification Window does not terminate/close, December 2013. Last accessed: 2014-06-28. URL: <https://code.google.com/p/yawl/issues/detail?id=491>.
- [FA14a] C. Freihoff and M. Adams. YAWL Bug Report Discussion: Issue 504 - leaving optional textfield empty leads task into nirvana with error message, February 2014. Last accessed: 2014-06-28. URL: <https://code.google.com/p/yawl/issues/detail?id=504>.
- [FA14b] C. Freihoff and M. Adams. YAWL Bug Report Discussion: Issue 512 - Some SpecificationData Getters allways return null, April 2014. Last accessed: 2014-06-28. URL: <https://code.google.com/p/yawl/issues/detail?id=512>.
- [FA14c] C. Freihoff and M. Adams. YAWL Bug Report Discussion: Issue 517 - User privileges are not taken by participant when using external resourcesDataSource, May 2014. Last accessed: 2014-06-28. URL: <https://code.google.com/p/yawl/issues/detail?id=517>.
- [FA14d] C. Freihoff and M. Adams. YAWL Bug Report Discussion: Issue 519 - YAWL looses Workflow after Database Foreign Key Violation, June 2014. Last accessed: 2014-06-28. URL: <https://code.google.com/p/yawl/issues/detail?id=519>.
- [FA14e] C. Freihoff and M. Adams. YAWL Bug Report Discussion: Issue 523 - Editor 3.0 allows no explicite mappings from primitive to complex datatype (worked in 2.3.5), June 2014. Last accessed: 2014-06-28. URL: <https://code.google.com/p/yawl/issues/detail?id=523>.

Bibliography

- [FA14f] C. Freihoff and M. Adams. YAWL Bug Report Discussion: Issue 524 - Editor 3.0 creates unwanted out-mapping for input only variable, June 2014. Last accessed: 2014-06-28. URL: <https://code.google.com/p/yawl/issues/detail?id=524>.
- [Got09] F. Gottschalk. *Configurable Process Models*. PhD thesis, Eindhoven University of Technology, The Netherlands, December 2009. URL: <http://alexandria.tue.nl/extra2/200613090.pdf>.
- [Gro08] B. Groene. How to communicate architecture - Technical Architecture Modeling at SAP (part 1), January 2008. Last accessed: 2014-06-28. URL: <http://scn.sap.com/people/bernhard.groene/blog>.
- [Grö13] M. Grönroos. *Book of Vaadin: Vaadin 7 Edition - 2nd Revision*. Vaadin Limited, 2013. URL: <https://vaadin.com/download/book-of-vaadin/vaadin-7/pdf/book-of-vaadin.pdf>.
- [HvdAAR10] A.H.M. Hofstede, W. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2010. URL: <http://www.springer.com/computer+science/database+management+%26+information+retrieval/book/978-3-642-03120-5>.
- [lif14] Portal Properties, March 2014. Last accessed: 2014-06-28. URL: <http://docs.lifera.com/portal/6.2/propertiesdoc/portal.properties.html>.
- [Mor14] D. Morosjuk. Integration von Web-Applikations-Entwicklungsumgebungen auf der Persistenz-Ebene am Beispiel von YAWL und Liferay. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences, 2014.
- [OM03] E. Ort and B. Mehta. Java Architecture for XML Binding (JAXB), March 2003. Last accessed: 2014-06-28. URL: <http://www.oracle.com/technetwork/articles/javase/index-140168.html>.
- [Ros13] M. Roskosch. Konzeption und Implementierung eines Configuration-Management-Systems mit mobilen Workflows. Master's thesis, Bonn-Rhein-Sieg University of Applied Sciences, 2013.
- [RW12] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. SpringerLink : Bücher. Springer, 2012. URL: <http://www.springerlink.com/content/978-3-642-30409-5>.
- [Sam13] K. Sampath. How-to clear liferay cache, May 2013. Last accessed: 2014-06-28. URL: <http://www.liferay.com/web/kamesh.sampath/blog/-/blogs/how-to-clear-liferay-cache>.
- [Sar12] A. Sarin. *Portlets in Action*. Manning Pubs Co Series. Manning Publications Company, 2012. URL: <http://books.google.de/books?id=P0qTSQAACAAJ>.
- [SK12] R.J. Sezov and B. Kim. *Liferay in Action: The Official Guide to Liferay Portal Development*. In Action Series. Manning Publications Company, 2012. URL: <http://books.google.de/books?id=xIA2bwAACAAJ>.

Bibliography

- [Val09] A. Valikov. Hyperjaxb3, March 2009. Last accessed: 2014-06-28. URL: <http://xircles.codehaus.org/projects/hyperjaxb3>.
- [vdAtH05] W. M. P. van der Aalst and A. H. M. ter Hofstede. Yawl: Yet another workflow language. *Inf. Syst.*, 30(4):245–275, June 2005. URL: <http://dx.doi.org/10.1016/j.is.2004.02.002>, doi:10.1016/j.is.2004.02.002.

List of Tables

2.1. Liferay's organizational model	14
2.2. YAWL's organizational model	15
2.3. Examination environment components summary	22
3.1. Sample organizational structure	31
3.2. Changes in Resource Service web.xml	55
3.3. Reported and solved relevant YAWL bugs	71
B.1. Setup web.xml changes	86

List of Figures

1.1. Relationships between process and workflow	3
1.2. Relationships between build time and runtime	4
1.3. Basic components of a WMS	4
1.4. Basic components of a WMS	6
2.1. Mapping of organizational data	14
2.2. Codelet operation schema	16
2.3. Hyperjaxb3 operation schema	18
2.4. Portal infrastructure	19
2.5. Vaadin architecture	21
2.6. TAM/FMC block diagram components	23
2.7. Expected overall architecture	24
3.1. Liferay welcome page	28
3.2. Deployment of YAWL core services	30
3.3. YAWL Resource Service login	30
3.4. Sample workflow	35
3.5. Entity class generation from XSD	36
3.6. Persistence Layer Deployment to YAWL	42
3.7. Persistence codelets	49
3.8. Matching variables in task and codelet	50
3.9. YAWL Resource Service default style	53
3.10. YAWL Resource Service adjusted style	53
3.11. YAWL form default and adjusted style	54
3.12. Workqueue portlet in view mode	60
3.13. Three alternatives to work on a task	61
3.14. Workitem portlet displaying YAWL form	61
3.15. Workqueue portlet and workitem portlet	62
3.16. Data overview portlet with offered actions	62
3.17. Create a new employee data set	63
3.18. Upper and lower Section of data details portlet	64
3.19. Vaadin control panel	70
3.20. Final overall architecture	72

Listings

2.1. Sample XML schema definition	10
2.2. Resource Service web.xml	11
3.1. portal-ext.properties	26
3.2. setenv.sh	28
3.3. hibernate.properties	29
3.4. employee.xsd	32
3.5. persistence.xml	36
3.6. ObjectFactory.java	37
3.7. persistence.properties	38
3.8. EmployeeAdapter.java	38
3.9. EmployeeXmlAdapter.java	40
3.10. LiferayPortalOrgdata.java	43
3.11. LiferayResourcesAdapter.java	45
3.12. EmployeeReaderCodelet.execute(...)	48
3.13. EmployeeUpdaterCodelet.execute(...)	48
3.14. CreateCredentialsCodelet.java	50
3.15. Generic portlet body	57
3.16. YawlWorkqueueAdapter.java	65
3.17. liferay-display.xml	67
3.18. liferay-portlet.xml	68
3.19. portlet.xml	68

A. Contents of the Attached Data Carrier

A.1. Root Folder

Contains the bachelor thesis document and the bachelor thesis exposé in PDF format.

A.2. Literature

Contains the literature used in this elaboration as PDF files.

A.2.1. Online Backups

Online sources have been downloaded and stored in PDF format.

A.3. Illustrations

All relevant illustrations and diagrams created for this thesis are stored in this folder.

A.4. Software

A.4.1. LiferayAddons

Contains files to configure Liferay and to prepare the portal for Vaadin applications.

A.4.2. PersistenceLayer

The first of three Maven projects represents the persistence layer including Hyperjaxb3, data access objects and generated entity classes.

A.4.3. Portlets

In this folder, the second Maven project can be found. It contains the implementation of the portlet application.

A.4.4. Prepared Bundle

Contains a prepared bundle which already contains Liferay Portal, the YAWL core services, the YAWL extensions, the persistence layer and several configurations. Only the installation of the Vaadin control panel, Vaadin version upgrade and widgetset compilation are required before the portlets can be deployed.

A.4.5. ResourceServiceLibs

Provides all necessary dependencies for the use of the persistence layer in the YAWL WMS.

A.4.6. Specifications

The sample workflow and the XML schema defined business object are stored in this folder. The business object definition is available in two versions (for HyperjaxB and for YAWL). The version for YAWL is already included in the workflow specification.

A.4.7. YawlCoreServices

Contains the YAWL core services in a current version. Tested and working latest builds are already included as well as configurations and extensions.

A.4.8. YawlExtensions

The third Maven project contains the extensions for the Resource Service. These are the organizational data import, the codelets and the CSS file.

B. Installation Manuals

B.1. Download URLs

1. Liferay Portal
<http://www.liferay.com/downloads/liferay-portal/available-releases>
2. YAWL core Services, YAWL library JARs for development
<http://sourceforge.net/projects/yawl/files/YAWL%20Engine/Release%203.0beta/>
3. YAWL latest builds
<http://www.yawlfoundation.org/pages/resources/latestbuilds.html>
4. YAWL editor 2.3.5
<http://sourceforge.net/projects/yawl/files/YAWL%20Editor/Release%202.3.5/>
5. Hyperjaxb3
<http://confluence.highsource.org/display/HJ3/Downloads>
6. Vaadin addons
<https://vaadin.com/directory>
7. This document and the associated source code, support in terms of the subject of this elaboration
<http://www.freihoff.org>

B.2. Expert Setup

This manual describes the setup from scratch by downloading the components from the manufacturers pages. Both YAWL and Liferay are developed further. So, it may be that old bugs return or new bugs find their way into the software. Be aware that only the prepared bundle contains versions which have been tested regarding their interaction with each other and the developed plugins/applications. Hence, the expert setup instructions are for developers only, additional info can be found in section B.4 below.

If You prefer the (recommended) save and easy way, use the prepared bundle from the provided data carrier or download (folder /Software/PreparedBundle) and let out the expert setup.

1. Provide a system with a current performance (at least 2CPU and 8GB RAM are recommended), Java 7 runtime environment and PostgreSQL 9.1 or newer.
2. Create 3 database users(passwords): liferay(liferay), yawl(yawl), personnel(personnel).
3. Create 3 databases with the same names and owned by the corresponding user
4. Download and extract the Liferay Tomcat bundle to a folder of choice (e.g., /opt)

B. Installation Manuals

5. Navigate to the bin folder of the Tomcat (e.g., /opt/lr622yawl3/tomcat-7.0.42/bin)
6. Open the setenv.sh (setenv.bat for windows) and set -Xmx4096m, -XX:MaxPermSize=384m and -Duser.timezone=Europe/Berlin (or Your timezone).
7. Download the YAWL core services and replace the respective ones by their latest builds (enterprise edition).
8. Unpack the YAWL core service WARs so that they are present as single folders (works with winrar).
9. Find all hibernate.properties (3-4, depending on version) and change the database settings corresponding to the configuration above (database: yawl, username/password: yawl).
10. Copy the core services to the webapp folder of the bundled tomcat (e.g., /opt/lr622yawl3/tomcat-7.0.42/webapps).
11. Open the attached data carrier (or download and extract the corresponding zip file if data carrier not available).
12. Update the YAWL-related dependencies in the YawlExtensions and Portlets projects. Because newer dependency jars may have the same names and versions as the current ones, clean up Your local maven repository. Then clean and build the projects.
13. Copy the folder /Software/YawlExtensions/target/classes/org to /resourceService/WEB-INF/classes (integrate in existing folder).
14. Copy the file /Software/YawlExtensions/target/classes/styleSheet.css to /resourceService/resources (replace existing file)
15. Copy all files from /Software/ResourceServiceLibs to /resourceService/WEB-INF/lib
16. Copy /Software/PersistenceLayer/target/persistenceLayer-1.0.jar to /resourceService/WEB-INF/lib
17. Open the web.xml file in /resourceService/WEB-INF and change the parameters according to the following table.

Table B.1.: Setup web.xml changes

Parameter name	Default value	New value
OrgDataSource	HibernateImpl	LiferayPortalOrgdata
ExternalUserAuthentication	false	true
OrgDataRefreshRate	-1 (disabled)	1 (one minute)

18. Copy the JARs in folder /Software/LiferayAddons to the lib folder of the root web application in Your Liferay Tomcat bundle (e.g., /opt/lr622yawl3/tomcat-7.0.42/webapps/ROOT/WEB-INF/lib). The original ehcache.jar has to be overwritten to fix a Liferay bug.

19. Copy the file /Software/LiferayAddons/portal-ext.properties to the root folder of Your Liferay Tomcat bundle (e.g., /opt/lr622yawl3).
20. Continue with step 5 in the following section.

B.3. Prepared Bundle Setup

1. Provide a system with a current performance (at least 2CPU and 8GB RAM are recommended), Java 7 runtime environment and PostgreSQL 9.1 or newer.
2. Create 3 database users(passwords): liferay(liferay), yawl(yawl), personnel(personnel).
3. Create 3 databases with the same names and owned by the corresponding user
4. Download and extract the prepared bundle to a folder of choice (e.g., /opt). Note: Due to a possible bug in the respective YAWL build, the path must not contain any whitespaces or extended UTF characters.
5. Open the portal-ext.properties file in the root folder of the bundle and adjust Your personal settings (e.g., admin user, company).
6. Start the Tomcat by running the startup.sh (startup.bat for windows) in the Tomcat bin folder, the startup will take a while. If an error message occurs on Linux, review rights and folder ownerships (sudo should always work). When server is up, the portal can be accessed on port 8080 on Your machine (e.g., <http://localhost:8080>). Check, if You can login with Your settings.
7. The Resource Service can be accessed via its path corresponding to the web application folder name (e.g., <http://localhost:8080/resourceService>). Administrative work like the upload of workflow specifications has to be done there. Log in as YAWL default administrator ("admin", pw: "YAWL") or as Your administrative Liferay user specified in the portal-ext.properties. Upload the provided sample workflow (specifications/CheckInEmployee2.yawl) there.
8. Copy the /Software/LiferayAddons/liferay-vaadin-plugin.war to the deploy folder of Your Liferay Tomcat bundle (e.g., /opt/lr622yawl3/deploy) and wait for the deployment to finish. If problems occur (e.g., the deployments starts but does not finish), shut down the tomcat while copying the WAR file into the deploy directory.
9. When logged in with administrative rights, go to the portal's control panel and open the Vaadin configuration (last entry in configuration column).
10. There, change the version to 7.1.8. The output console should give a success feedback.
11. Select the shown addons (at least QRCode and Refresher should be selectable) and click the button to compile the widgetset. Six permutations will be compiled, this may take a few minutes depending on system performance.
12. Add the sample organizational structure (cf. table in section 3.2) to the portal. For testing purposes it is enough to add the regular roles and assign them to Your default user.

13. Now, Liferay is ready for deploying Vaadin portlets. Copy the file `/Software/Portlets/target/YAWL-portlets-1.0.war` to the deploy folder. If problems occur, act as described in step 8.
14. Finally, the portlets can be added to pages of Your choice by the add applications dialog available for administrative users. To work properly, the Workitem Portlet has to be placed on the same page as the Workqueue Portlet. The portlets for data overview and data details must also be placed on one page.
15. The basic settings of the portlet applications can be adjusted via the `portletconfig.properties` file which can be found in the respective webapp folder of Your tomcat (e.g., `tomcat-7.0.42/webapps/YAWL-portlets-1.0/WEB-INF/classes`). Additionally, some settings can be done in the preferences of the respective portlets (only valid for the current session).
16. To learn more about the administration of portal and portlets, please refer to the Liferay documentation.

B.4. Development Setup

There are three Maven projects, which are

- PersistenceLayer (containing Hyperjaxb3 generator, generated entities, data access objects)
- YawlExtensions (containing codelets and external orgdata plugin)
- Portlets

You should be familiar with Maven development. The PersistenceLayer serves as dependency for the other two projects and must therefore be opened in the IDE when working on Portlets or YawlExtensions. This works with Netbeans but has not been tested with other IDEs. Dependencies which are not available in the Maven repository are stored internal repositories inside the projects (see the `pom.xml` files for details).

In the Portlets project code, you will find an additional portlet (the RestPortlet) which serves for testing. You can try out new components or anything else in there. The contained REST service implementation is not needed by the rest of the project and can be removed. This portlet deploys together with the other ones and can be added to the portal the same way. Apart from these notes, there is nothing special to be considered.