h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

camunda
The Business Process Company

Integration of Content into Enterprise Applications - Analysis, Conceptualization and Prototypical Implementation using Java EE and JCR

Master Thesis
Master of Science in Computer Science

Darmstadt University of Applied Sciences

Hendrik Beck

1st Supervisor: Prof. Dr. Bernhard Humm
2nd Supervisor: Prof. Dr. Inge Schestag

Started on: June 16th 2007
Date of submission: January 16th 2008

Declaration of Academic Honesty

I hereby declare to have written this thesis on my own, having used only means and resources explicitly listed here.

Hendrik Beck
Darmstadt, January 16th 2008

Thank you

Prof. Dr. Bernhard Humm, Darmstadt University, for interesting discussions and for a helpful and pleasant supervision.

Bernd, for inspiring discussions, useful hints and support during this time and for your valuable feedback.

My sister Melanie, for believing in me - even after having read through the whole thesis. And for your feedback after having done that.

My family, again for making all this possible. Thank you for everything.

Thu', for being there, for keeping up my motivation, for constantly believing in me and for taking care.

Abstract

"Content", defined as data that is mostly semi-structured or unstructured, is becoming more and more important in business contexts. In enterprise applications, per se operating with fully structured data sets, content-related features like full text searches are costly to implement.

This thesis investigates different types of data as well as their utilization in enterprise applications and content applications. This is the foundation for further elaboration on a decision process. This process aims to find a proper implementation model based on the types of data required to be processed, the required content features as well as the expected integration complexity of enterprise data and content data. It then leads to enterprise applications or content applications in case of homogenous sets of data or to hybrid solutions in case of heterogeneous sets of data.

Subsequently, hybrid systems are under investigation. The integration complexity is of particular interest, being the largest cost driver in this model of implementation. A framework on top of Java EE 5 and JCR has been developed that integrates enterprise and content data on application level. This approach is called "Total Integration" and makes further, manual integration tasks obsolete. As the concerns of the integration have been separated out of the actual enterprise application, initial development efforts as well as maintenance costs are also reduced.

Kurzfassung

Der Begriff "Content", definiert als Daten die größtenteils semi-strukturiert oder unstrukturiert vorliegen und mit vollstrukturiert ausgerichteten Systemen nur unzureichend verarbeitet werden können, erlangt im Unternehmenskontext immer größeren Stellenwert. In Unternehmensanwendungen auf Basis von z.B. Java EE oder .NET lassen sich Content-verarbeitende Funktionalitäten wie Volltextsuchen nur aufwändig implementieren.

Die vorliegende Arbeit untersucht die unterschiedlichen Typen von Daten sowie deren Einsatz in Unternehmensanwendungen und Content-Anwendungen. Darauf aufbauend wurde ein Entscheidungsprozess entwickelt, der es erlaubt, unter Einbeziehung der zu verarbeitenden Daten, der benötigten Funktionalitäten zur Verarbeitung von Content sowie der zu erwartenden Integrationskomplexität von Unternehmensdaten und Content-Daten eine fundierte und dokumentierte Entscheidung über ein geeignetes Implementierungsmodell zu finden. Dies führt zu den Varianten reiner Unternehmens- bzw. Content-Anwendungen bei einer homogenen Datenbasis oder einer Hybrid-Anwendung bei heterogenen Daten.

In Hybrid-Systemen, die im weiteren Verlauf von besonderer Bedeutung sind, spielt die Integrationskomplexität eine große Rolle, da sie den hauptsächlichen Kostentreiber in diesem Implementierungsmodell darstellt. Es wurde ein Framework auf Basis von Java EE 5 und JCR entwickelt, das Unternehmensdaten und Content-Daten aus zwei verschiedenen Systemen auf Anwendungsebene integriert. Der "Total Integration" genannte Ansatz macht zusätzliche manuelle Integrationsaufgaben obsolet. Da die Belange der Integration aus der eigentlichen Zielanwendung extrahiert werden, werden initialer Entwicklungsaufwand und Wartungskosten zusätzlich reduziert.

Contents

Chapter 1

Introduction

1.1 Motivation

After *Content* has been a field of research for a long time it first became popular with data-centric web applications and web content management systems. In recent years its significance and awareness is rising in the context of enterprise environments. Especially its characteristic of unstructured and semi-structured data, roughly speaking data without recognizable or predictable structure like videos, Office documents or freely written text, is applicable for a huge quantities of information existing in organizations. According to Gartner as much as 80% or more of company-wide existing information relates to those categories of data[1]. The discipline of Enterprise Content Management provides definitions of terms and delivers solutions to effectively manage that kind of data. Enterprise Content Integration is working on strategies and concepts to create common integrated data repositories on high organizational levels.

In the context of enterprise applications nonetheless, content is traditionally playing a minor role. These applications work with fully structured data, often specified through entity objects that are designed according to object-oriented paradigms and are then being mapped into relational databases. This approach per se lacks of support for unstructured and semi-structured data. Usually content data and documents are stored as text or binary large objects (BLOB), structure and semantics are mostly ignored. Certain use-cases in conjunction with content, e.g. versioning, internationalization, full-text searches or inexact queries aren't part of standard persistence products. If necessary, those features have to be modelled and implemented into applications on a case-per-case basis and tend to become cost-drivers despite their actual triviality. Due to the fact, that specialized content management applications easily fulfill those requirements, this often leads to parallel usage of both kinds of applications in order to cover the requirements. This approach will be called *hybrid implementation*.

This leads to a seperation of data into one of the two systems even though from a business point of view they were identified as belonging to the very same entities. Afterwards, portions of the data often have to be re-integrated in order to allow simultaneous processing in target applications. Once originated to reduce content-related costs, the complexity of this re-integration arises as the new cost driver in those hybrid implementations.

This thesis tries to formalize this process of finding the appropriate model of implementation. In case of hybrid implementations this process is also intended to support the decision on how to separate a given set of data into two different systems. It also evaluates the main cost drivers for different implementation models. It then takes a closer look at hybrid imple-

[1] See [RK05].

mentation. Based on desired cost reductions a prototypical solution will be developed that integrates enterprise data and content data and reaches significant cost reductions.

1.2 Scope, Goals and Contribution

The scope of this thesis are:

- To examine the characteristics of fully structured, semi-structured and unstructured data in enterprise environments and their implications on implementation models. Subsequently, a decision process model is introduced that supports architects and developers in determining proper architectural choices based on the actual data that are required to be processed in the target system.
- To investigate the cost behavior of the identified implementation models in order to develop strategies to reduce costs of implementations.
- To conceptualize a solution that performs a generic integration of data stored in two systems. The principles behind this solution should be applicable for a wide range of different standards and products in the world of enterprise and content applications.
- To develop a prototypical application that implements the concepts described beforehands.

The concrete goals expected as outcomes of this thesis are:

- A prototypical decision process that leads to implementation models based on given entity data required to be processed by an application.
- In case the outcome of the process leads to a hybrid implementation model, this process should also support the decision, which system to store particular data in.
- The main cost drivers of the different implementation models should be identified. This should help to better understand the rationales for one or another model.
- A major focus of this thesis also lies on hybrid implementation models. Thus, based on the main cost drivers a concept should be developed on how to significantly reduce costs in hybrid implementation models. This concept should be prototypically implemented.
- A small-scale example application should serve as a playground for demonstrating the decision process as well as the implemented solution.

1.3 How to read this thesis

Chapter 2 of this thesis introduces the basic concepts behind enterprise application frameworks, gives a market overview and explains how the basic concepts are being realized in Java Enterprise Edition 5.

Chapter 3 introduces definitions around content in enterprise environments. Besides that terminology, it also gives concrete use-cases that occur in conjunction with content. Finally it gives a market overview and shows how the standard JSR-170 addresses the general concepts of content management in a standardized model.

Chapter 4 compares the different types of data processed in enterprise applications and content applications as well as differences between the actual systems themself. Based on that

it explains how this leads to different implementation models in a given project situation. A decision process model is presented to support architects and developers in determining an appropriate model of implementation.

Chapter 5 describes in detail the integration framework that has been developed throughout this thesis. It introduces a user story and example application, explains technology-independets concepts of the thesis, describes the implementation and shows how it can be integrated in the example application.

Chapter 6 evaluates the concepts as well as the prototypical application in accordance with the requirements stated earlier in this chapter. It shows that all the goals of this thesis have been reached. It also shows how the concepts and the experience taken from the prototype can be used to determine the general value in other environments than the one of this thesis.

Chapter 7 summarizes the work of this thesis, gives a high-level conclusion and outlook for further fields of investigation.

The **appendix** contains resources of interest for this thesis such as source code, diagrams and links. Many of them are referenced within the text.

1.4 Typographical conventions

The following conventions are used throughout thesis:

Java class names Java classes and class names are written in *italic* typeface. As full-qualified Java names tend to get very long due to deep package hierarchies, the first four packages are usually abbreviated. Instead of *com.camunda.research.jcr.cef.impl.ocm.Content* the abbreviated spelling *c.c.r.j.cef.impl.ocm.Content* will usually be used.

Source code Source code is written in `Courier` typeface, mostly in separated boxes. Please note that source code might not be directly executable due to omissions of irrelevant parts where applicable, e.g. exeception handling. In those cases, omissions are intended to clarify the essential parts of the source code and thus enhancing understandability of the examples.

1.5 Applications and source code

Applications and source code developed throughout the thesis are available from the autor.

Chapter 2

Enterprise Application Frameworks

This chapter introduces enterprise applications and enterprise application frameworks in the contex of this thesis. After a rough and general overview enterprise applications in general are introduced. After that a brief market overview is given on frameworks that are seen as enterprise application frameworks. Finally Java EE 5 as a popular and modern enterprise application framework is introduced. It is also explained how the integral parts of enterprise application frameworks are specifically realized in Java EE 5.

2.1 Terminology

2.1.1 Enterprise Applications

A definition of the term *Enterprise Application* will be required for this work. It's hard though to identify a common clear definition from available literate. Inspired by various sources[1] the author will give his own definition considerered to be valid throughout this thesis:

> *An Enterprise Application is a piece of software developed and used mostly for business purposes of one or several business domains or whole organizations. It stores data, processes data within rather complex business logic implementations and may have user interfaces of some kind. It tends to be large, complex, distributed and embedded into a wider scope of applications within an IT landscape. Hence it may also be required to integrate several other ressources or applications. The services provided by this application are usually mission-critical to the organization it is run by. This implies that strong technical requirements and constraints have to be met in order to provide the reliability, performance and scalability.*

It has to be noted that this is not intended to give a common, unambigious definition of the term. It has rather the purpose to narrow done the scope of this thesis which applies to applications that are similar to the ones defined above.

2.1.2 Enterprise Application Frameworks

Based on the given definition of an enterprise application, the term *Enterprise Application Framework* is defined as follows:

> *An Enterprise Application Framework is a piece of software that supports application developers in reaching technical and business requirements for an enterprise applications. Therefore, it usually defines technical services that implement*

[1] See e.g. *http://www.theserverside.com/discussions/thread.tss?thread_id=38042*, [MB07], [GH03] [MF02].

or help to implement certain technological features like transactions, persistence or high-availability concepts. To be able to achieve this complex set of features and requirements, it usually also specifies a certain runtime environment. The two enterprise application frameworks that are considered to be the closest to the definition required in this thesis are Java EE and .NET.

2.2 Conceptual Overview

Enterprise application frameworks generally support in developing and running enterprise applications. The requirements commonly existing in modern, large enterprise applications lead to certain concepts that are usually covered by enterprise application frameworks. [MB07] for example is giving the following requirements typically to be fulfilled by enterprise application frameworks:

Back-end integration Back-end systems for data storage must be integrated into the application, most typically a database system.

Persistence It must be possible to persist enterprise data in a proper way. A popular approach is to map object-orientedly designed entity objects to the relational model in order to store them in a relational database.

Consistency of data For complex operations with multiple individual steps, support of transactional processing is required. See also section 2.3.5 for a detailed explanation.

Client access Since enterprise applications tend to be large, it must be ensured that multiple concurrent client accesses are possible. Furthermore, authentication and authorization are also important issues. Furthermore, they must also be able to access the same set of data concurrently.

Performance, scalability, availability These non-functional requirements also have to be fulfilled. Especially in applications that tend to become large, complex and distributed, these issues become very important.

Software design The architecture as well as the principal paradigms must ensure that software components are maintainable and portable. Appropriate layering of applications and modularizations are widely accepted tools used to reach these goals and have to be reflected by the framework.

An integral part that is often not covered by literature about enterprise application frameworks is the the existance of a concept to implement *business logic*. Also, it is difficult to find dedicated definitions for the term itself. In this thesis nevertheless, it is important to outline that an enterprise application basically processes data. The way the data is processed is implemented using programming languages and consists of algorithms appropriate to define the required behaviour.

From the business logic point of view, most of the other integral parts of an enterprise application are merely supporting this data processing. User interfaces, for examples, are used to involve the user into the process of retrieving data from the user that have to be processed and to present results of this processing. Databases are used to store data that has been processed or may eventually be processed at a later time. Requirements like the performance are used to ensure that data processing operations don't exceed timeframes constrained by external business factors. In conclusion, all components are working together in order to

provide services, but in its core enterprise applications are processing data in way that is defined by its business logic.

In the established three-tier application model[2], business logic is located on the middle tier. This is often referred to as the *business logic tier*. Business logic in modern enterprise applications is usually developed based on the concepts of object-oriented design, object-oriented programming languages and is usually component-based. An Example would be EJB, the component model of Java EE.

In this thesis the business logic is essential in comparison with content management applications, introduced in chapter 3. The focus in those systems is to actually store data and provide data-oriented services to clients. Enterprise applications are focussing on business logic and hence they are used whenever data has to be processed in complex ways. This requirement to process data will become one of the influencing factors of the decision process introduced in chapter 4.

2.3 Data in Enterprise Applications

2.3.1 Structuredness of Enterprise Data

The characteristic of persistent data that is most important in the context of this thesis is the *structuredness*. Enterprise applications are usually designed to work with *fully structured data*.[3]

This not only influences the way data are finally stored, e.g. in a relational database. It also influences how data are modelled on application level, e.g. in object-oriented entities. The structure of this kind of data has the following characteristics:[4]

Pre-defined The structure is defined a-priori. In databases, schemas define tables, fields, data types and the like beforehands. In object-oriented programming languages, classes together with its attributes are defined during development and fixed at compile time.

Explicit The schema defines the structure ultimately. The actual values stored in it do not contain any information about the structure. Relational databases as well as object-oriented programming languages store schema information once at a central place, tables and table rows itself as well as object instances only hold the actual values.

Regular For each instance of a defined entity the structure remains the same. Every row of a table contains the same fields as well as every instance of a class contains the same attributes. Values on the other hand can still be null, i.e. they do not contain anything. But still the structure reserves the space for the actual value even if it is null.

Complete This also means that fields are never ommitted. Every instance contains the exact same fields as defined within the structure.

Immutable The structure is immutable, i.e. it never changes. Unless, of course, the structure is changed. But as the structure is kept centrally, the change would also have to be done in this central structure made from outside. This would lead to a change in all data stored with this structure.

[2] See [DK75].

[3] It is often often referred to as just *"'structured data"'*. But for the sake of distinction to *semi-structured* and *unstructured* data, here it will be called *"'fully structured data"'*.

[4] See [Abi97].

Constraining The structure constrains data, ensures type safety and, generally speaking, protects the data from being used in the wrong way.

These characterstics of fully-structured data has two major consequences, that are also of interest in the context of this thesis:

1. **The structure is predictable** As the structure is completely defined and known beforehands, applications can make assumptions about the data. Similarly, strongly-typed programming languages also make use of this characteristic. At development time it can predicted which classes are available and which attributes they contain.

2. **The structure is not very flexible** Structures have been known completely beforehands. Dynamic structures are not possible per se. They are always reached by different workarounds.

2.3.2 Relational Databases

Relational databases, implementing the relational model, are by far the most popular way to store fully structured data in today's IT infrastructures.[5] At different times the relational model competed, and succeeded, against other database models, most notably the hierarchical model, the network model and the object model. Its innovative concept introduced set-oriented, flat and relation-based data management and a mathematical yet simple foundation.[6]

Relational databases contain the following major elements:

Tables Tables are entities that are intended to hold values. In the relational model, tables are called *relations*. A relational database consists of a collection of tables.

Fields Tables may contain fields. These fields are of a certain data type, like string, integer or boolean, and hold a value of this type. Due to the usual visualization of tables and fields, fields are often universally addressed to as *columns*. One entry of a table is often referred to as a *row*.

Primary Keys Each table might have one or more fields declared as primary keys. The values of these primary key fields must be unique within an instance of this table. Rows can unambigiously be identified using primary keys.

References Two tables can be connected with each other. A field of one table, the foreign key, must be connected with the primary key of another table. These connections are called references.

Besides tables, references are used to create structures and express connections within relational databases. This was a major step off from hierarchical database models, where connections between entities had to be expressed by hierarchical structures whereas the relational model allows to modell small, simple and independent entities. This can be compared with classes in object-oriented programming. They are per se independent but may be connected with other classes e.g. via associations.

Tables and their containing fields are designed a-priopri. This is done by defining a *schema*. Schemas are following the characteristics of fully-structured data given in the previous section.

[5] Both were first developed and published by Edgar Codd. See [Cod70] and [Cod90].

[6] Information about relational databases were taken from [AK04], [AH00] and [RE07].

2.3.3 CRUD

The acronym is extended to *Create-Read-Update-Delete*. It denotes the four possible basic persistence operations. A system offering or implementing all these persistence operations is considered to be complete, i.e. to provide a complete set to perform any required operation. The pattern is in common usage throughout the literature.

2.3.4 SQL

SQL, the *Structured Query Language*, was developed in 1974 by Donald D. Chamberlin and Raymond F. Boyce. It was designed as a data management and retrieval language for relational databases. SQL is considered to machine-readable as well as intuitive and easy to read for humans due to its obvious lingual similarity to the English language. SQL is a descriptive, interpreted language. SQL was standardized by ANSI[7] in 1986 and by ISO[8] in 1987 with standards also exisiting from 1992, 1999 and 2003. Furthermore, many vendor- and database-specific dialects are in use. SQL is divided in four major groups of statements:[9]

Data definition These statements can be used for data modelling purposes like creating, altering and dropping tables or adding fields to tables.

Query They are used to select exisiting data within tables or referenced data existing across tables.

Data manipulation These are used for tasks like adding, updating or deleting data.

Data control Use to grant and revoke user's access right in order to achieve authorization for existing data.

2.3.5 Transactional Processing of Data

Transactions are used to ensure that complex operations are being executed correctly. The larger the number of individual steps is, the bigger the amount of time is the whole operations needs to be finished and the more distributed in terms of different phyical places the operation is, the more complex the operation will be. And hence the more problems arise. Transactions are addressing these problems. The term transaction refers to the actual operation that has to be perfomed.

The ACID model describes requirements transactions have to fulfil on execution time in order to be called a sucessful transaction:[10]

Atomicity The transaction must be executed completely or not at all. This means that if one operation within the transaction fails and is not executed successfully, then all the other operations, if already finished or not even begun, must not be executed. To be able to archive that, the terms *commit* and *rollback* are important and will be explained afterwards. This characteristic of ACID is of particular interest in the context of this thesis as the hybrid persistence developed throughout the thesis adds another operation to every single persistence operation.

[7] ANSI, the American National Standards Institute, see *http://www.ansi.org*.

[8] ISO, the International Organization for Standardization or Organisation Internationale de Normalisation, see *www.iso.org*.

[9] See e.g. [RE07], p.233.

[10] See [JG93], [PAB97] p.8.

Consistency Consistency of the underlying storage has to be maintained even in case of a failure.

Isolation All transactions are running isolated from other transactions concurrently executed. Executing transactions must not be affected from results or effects of other transactions.

Durability Any successful transaction leads to results, that will be persisted durable, e.g. written into the file system or stored into a database.

As soon as two or more operations have to be executed as transactions satisfying the ACID rules, the border of the transactions must be clear. Right before the first operation the transaction will be started. This is the *begin* of the transaction. This is done by a *Transaction Processing Monitor*[11]. The TP monitor doesn't participate in the actual transaction but observes and controls from outside. In case any operation during the execution of the transaction fails, none of the operations should be executed. Due to sequential-processing[12] it can happen that some of the operations have already been performed successfully. Those operations must be reverted so that they appear as not performed but still respect the ACID criteria. Due to that fact the terms *roll-back* and *commit* have been introduced, along with a *transient state* of operations. This means that transactions are being performed according to the following schema:

1. An application is starting a transaction in accordance with the transaction monitor.
2. After that all operations are being performed. Every single operation that is finished executing successfully still remains in a transient state because it might happen that the whole transaction might fail. This means that the results can still be reverted. Concrete implementations of this behaviour are up to particular components involved in the transaction.
3. If an operation fails, the whole transactions has to fail. This is called a roll-back of the transaction, along with roll-backs of each operation that has been executed so far.
4. If each operation completes successfully, then whole transaction succeeds. This is called a commit. Again, this leads to a commit of every single operation transforming their transient states to final persistent states.

In client-server applications transactions might not only be executed within the same server. They may also be initiated by local or remote clients or be executed if servers communicate with each other. Such involvement of machines on different network hosts is called a *distributed transaction*. Whereas the constraints remain the same as for local transactions the effort and difficulty level to implement and manage those transactions rises. Enterprise application frameworks provide support for transactions.

Transactions can also contain operations that are performing transactions themselves. This is usually referred to as a *nested transaction*. However, there are different strategies on how to react in such a situation. For example it is common, that transactions realize if there's an exisiting transaction available and "'join"' that transaction rather than initiating their own transaction. This decision is usually the responsibility of the developer and is done beforehands.

[11] Also referred to as *Transaction Processing System*, *Transaction Monitor*, *TP system* or *TP monitor*. See also[JG93] p.239.

[12] In most common programming languages and processor architectures, operations are performed sequentially despite a logical concurrency those operation are often regarded as.

2.4 Market Overview

This section introduces popular frameworks that provide application developers with integral features necessary for enterprise application to be developed efficiently and to be run properly.

2.4.1 Overview

Over the years many tools, applications, frameworks and standards evolved to address the main requirements for enterprise applications. It can be stated that the requirements don't change essentially over time. But solutions vary and have been evolved over time in order to provide application developers with ever more features, convenience and performance to realize their applications.

In early years, transactions have been the first aspects seriously addressed by frameworks supporting enterprise applications. The *Customer Information Control System*[13], developed by IBM, is a popular system supporting transactions. *Corba*[14], developed by the OMG[15], was a standard intended to allow components running of different platforms and written in different programming languages to communicate with each other. Such frameworks addressed only parts of the requirements necessary to develop and run enterprise applications. Later, frameworks were developed that support nearly every aspect of enterprise applications. They are intoduced in the following.

2.4.2 Java Enterprise Edition

Java Enterprise Edition[16] is a platform for developing and running client-server enterprise applications based on Java. First released in 1999, Java EE has become one of the most successful enterprise applications framework available. Java EE was also chosen to be the platform of choice in this thesis. Hence, section 2.5 will elaborate on details of Java EE.

2.4.3 .NET

Microsofts *.NET* platform platform was first introduced in 2000. The first version 1.0 was released later on in 2002. .NET is widely seen as the strongest competitor to the Java EE platform yet both platforms are sharing a lot of conceptual similarities. Differences can be seen in the number of available implementations, frameworks and tools due to Sun's more community-centric policies. Although there are ongoing discussions about advantages and disadvantages of both[17], the conceptual differences are marginal and technical details are rarely decisive factors in favour or against one of them. More information can be found e.g. in [Lib05].

[13] Abbr.: *CICS*. See e.g. *http://www-306.ibm.com/software/htp/cics/library*.
[14] See *http://www.omg.org/technology/documents/formal/corba_2.htm*.
[15] Abbr.: *Object Management Group*. See *http://www.omg.org*.
[16] Abbr.: *Java EE*, often referred to as *J2EE*.
[17] See e.g. *http://www.oreillynet.com/pub/a/oreilly/java/news/farley_0800.html* or *http://www.theserverside.com/tt/articles/article.tss?l=J2EE-vs-DOTNET*.

2.5 Java 5 Enterprise Edition

2.5.1 Overview

The *Java Enterprise Edition 5*[18] is a platform widely adopted to implement enterprise applications as introduced earlier in section 2.2. It has been developed under the Java Community Process[19] as JSR-244. The final release was published on May 11th 2006.[20]

It consists of specifications on how enterprise applications should be developed. This includes how different APIs and specifications are managed and wired together and conforms how application servers should be designed, the runtime environment for enterprise applications based on Java EE. Notable examples of API's included in Java EE 5 are:

- Enterprise Java Beans
- Transactions
- Java Naming and Directory Interface (JNDI)
- Java Messaging Service (JMS)

2.5.2 Business Logic

Java EE contains the specification of a component model called *Enterprise Java Beans*[21]. The version of the EJB specification included in Java EE 5 carries the version number 3 and was developed under the Java Community Process as JSR-220.[22] It was released on May 11th 2006. Physically EJB defines two different artifacts: a specification written in plain text and a set of Java API's.

EJB can be seen as a standardized way to write server-side components. Due to its conformity with the specification, components developed with EJB can be run in any compliant runtime environment, called the *EJB container*. EJB also specifies this EJB container, which is in practice usually part of an application server. This container is managing the components which allows to outsource many standardized tasks and responsibilites into the container. Examples for services provided by the container are lifecycle management, persistence or transactions. EJB aims to reduce implementation overhead for the application developer.

EJB provides an architecture to develop business logic in a standardized way using the Java programming language. EJB defines different types of so-called *beans*. These beans are generally adding information or behaviour to standard Java classes in order to allow the EJB container to manage them appropriately. Based on the use-case that has to be implemented, one of the following types provide the required behaviour.

Stateless Session Beans A session bean contains implementational logic and is intended to fulfill business requirements. Session beans can be accessed by other session beans or by clients directly. The EJB container is managing the lifecycle of this class, e.g. instantiation or pooling. This bean is *stateless* as clients are not allowed to rely on any inner state held by the session bean. As a technical consequence, instances of this type of bean are usually randomly assigned to any client making a request. Hence,

[18] Abbr.: *Java EE 5*.
[19] See *http://www.jcp.org*.
[20] The specification can be found in [(JC06b].
[21] Abbr.: *EJB*.
[22] The specification can be found in [(JC06a].

it is not guaranteed that a client request is consecutively processed by the very same instance. This type of bean is generally preferred due to lower management overhead in comparison with the stateful session bean explained afterwards. This type of bean can be used for any operations that do not have to carry an inner state.

Stateful Session Beans Stateful session beans are carrying an inner state. They are instantiated and assigned for exactly one client and are being destroyed afterwards without being used by another client.

Message-driven beans Message-driven beans are intended to be used in asychnrounous communication with client applications.

Entity beans Entitiy beans are containing data that is supposed to be persisted in a database. The EJB container is taking out high amounts of the work necessary to successfully store data in a database. The persistence aspect is covered in section 2.5.3.

2.5.3 Persistence

The specification of EJB 3 defines JPA, the Java Persistence API. It is a framework to manage, persist and query relational data. JPA was developed under the Java Specification Process as part of the specificiton process for EJB, JSR-220. It is widely recognized as an enourmous improvement on EJB 2.X regarding persistence issues. In the context of this thesis the most important difference to persistence mechanisms in earlier versions of EJB is that JPA is intended to only work with relational data sources. Other storage format such as XML databases or object-relational databases have been dropped in favour of relational databases. The most notable characteristics of JPA are given in the following:[23]

- JPA uses a POJO[24] model. Particularly, classes containing persistent data don't have to implement certain interfaces or the like.
- JPA entities are configured using an external deployment descriptor or embedded annotations[25]. Typcial configurations are to mark classes as persistent, to mark associated persistent objects or to alter the default configuration of JPA.
- Persistent objects can be detached from its logical connection to the persistent storage. Then, they can be used as transfer objects. Later on, they can be attached again in order to merge changes into the persistent state. This makes *Data Transfer Objects*, or *Value Objects*, obsolete. Later on, this characteristic is used by the framework, that is being developed.

Listing 2.1 shows a simple example Java class that holds persistent data. The annotations *@Entity*, *@Id* and *@OneToMany* have to be noted. They provide configuration values for JPA and they also change a normal Java class into a persistent Java class.

The annotation *@javax.persistence.Transient* is showing how JPA can be told not to persist a particular attribute. That way, attributes are ignored by JPA and will not be stored in the database.

[23] See [MB07], p.112.

[24] "'Plain Old Java Object"'. The term was coined by Martin Fowler et al., see *http://www.martinfowler.com/bliki/POJO.html*

[25] Annotations are introduced later in section 2.5.6.

Listing 2.1: Example JPA Entity Class

```
1  @Entity
2  public class Customer {
3
4    @Id
5    private String id;
6
7    private String name;
8
9    @OneToMany
10   private Order[] orders;
11
12   @javax.persistence.Transient
13   private String otherInformation;
14
15 }
```

2.5.4 Transactions

After section 2.3.5 introduced the concepts behind transactions in enterprise applications this section is looking at how transaction processing is realized in Java EE 5[26].

Transactions in Java EE 5 are part of the specification of Enterprise Java Beans version 3.0, that has been introduced in section 2.5.2. EJB 3 prescribes the usage of transactions according to JTA, the Java Transaction API specified in JSR-907[27]. JTA provides a standardized API to use transactions and provides an abstraction from concrete implementations of participants of transactions such as the transaction processing monitor. According to EJB 3, the application server serves as transaction process monitor and is thus responsible for creating and monitoring transactions.

Setting transactions

Java EE 5 provides support for different types of exceptions. They mostly vary in who is responsible for initiating the transaction.

Container-managed The application server is taking care of transaction initiating, rolling back and comitting transactions. Therefore the developer has the choice to decide whether certain methods or all methods of a class should be executed within a transaction. This can be done declaratively by using the deployment descriptor or annotations. It can be enabled by using the annotation *@javax.ejb.TransactionManagement()* and the value *javax.ejb.TransactionManagementTy*

Bean-managed The bean itself takes care of transaction handling programmatically. This can be enabled by using the annotation *@javax.ejb.TransactionManagement()* and the value *javax.ejb.TransactionManagementType.BEAN* and by inserting commands to manage the transaction. The resulting code basically looks like in listing 2.2.

Client-managed It is also supported that clients initialize transactions. This actually is not really a distinct type of transaction but rather a capability of the application server to take over existing transactions that were initiated by a client. In case of remote clients those transactions become distributed transactions.

[26] See also [(JC06a], chapter 13 "'Support for Transactions"' on page 315 and following.
[27] JTA's final release was published on November 6th 2002. For more details see [(JC02].

Listing 2.2: Example Source Code Transactions

```
1 utx.begin();
2 try {
3     doSomething();
4     doSomethingElse();
5     utx.commit();
6 } catch (Exception ex) {
7     utx.rollback():
8 }
```

Rolling back transactions

Certain errors occuring on execution of an operation within an exception should eventually lead to rolling back the complete transaction. This can basically be done in any component involved in the transaction by marking the transaction as having to be rolled back. Also, exceptions thrown by any participating component within a transaction eventually lead to a roll-back. According to EJB, checked exceptions dont't lead to roll-backs while unchecked exceptions do. Furthermore applications can set to roll-back or not to roll-back transactions explicitly by using the annotation *@javax.ejb.ApplicationException(rollback=true)*. Further details can be found in the specification of EJB 3.0[28]

2.5.5 Interceptors

Interceptors are a tool to customize and extend the behaviour of applications at runtime. They are defined by the EJB 3.0 specification.

Interceptors are Java objects that are implementing a piece of logic or behaviour.[29] This object is never specifically called by application code. Developers can declaratively tell the EJB container to execute this code whenever methods of session beans or message-driven beans are executed. Usually, a common behaviour applying for many methods or classes of an application are recommended to be extracted. This leads to simpler application code and to a separation of concerns[30]. It has also similarities with the concept of cross-cutting concerns and joint points in aspect-oriented programming.[31]

Listing 2.3 shows an example interceptor class *LoggingInterceptor*. This interceptor is intended to extract the concern of logging out of the actual application. The interceptor method is being called before the session bean call has been made. Within the interceptor method, this method call to the session bean can now be done. Before or after this call, other code like logging can be executed. The listing also shows how the interceptor is registered on the example session bean. It can either be registered on class level, for all methods of this class, or more specifically on method level. More features exisit, e.g. to make a more fine-grained selection on registered methods. Interceptors will be used in chapter 5 to separate the concern of activation out of the enterprise application. This way, a common behaviour for a set of methods can be achieved without altering the actual application.

[28] See [(JC06a], chapter 14 "'Exception Handling"' on page 355 and following.
[29] See [MB07], p.346.
[30] See [Dij82].
[31] See [Lad03], p.33.

Listing 2.3: Example Interceptor Class

```
1  public class LoggingInterceptor {
2
3    @AroundInvoke
4    public void log(InvocationContext invocation) {
5      logger.info("'We are right before actual method call!"');
6      invocation.proceed();
7      logger.info("'And now the method call is over!"');
8    }
9
10 }
11
12
13 @Stateless
14 @Interceptors(LoggingInterceptor.class)     // either here...
15 public class ExampleSessionBean {
16
17   @Interceptors(LoggingInterceptor.class)   // ...or here
18   public void doSomething() {
19     doSomethingMore();
20   }
21
22 }
```

2.5.6 Annotations

Annotations are a tool to add meta information to source code elements that has been introduced in Java version 5.[32] Annotations are used according to listing 2.4.

Listing 2.4: Java Annotations

```
1 @Annotation(paramater=parameterValue)
2 element (e.g. class, method, field ...)
```

The annotation itself is implemented as a Java type with the introduced type name *@interface*. They are widely used to enrich Java source code with meta information that can be read during run time. Annotations don't affect the actual byte code but rather are stored as additional information. This information can be retrieved by the Java reflection mechanism.

Many Java projects provide annotations as a way of configuration. Often XML and annotations are offered for configuration purposes at the same time, leaving it up to the developer which one to use. Java EE 5 is an example of a recently released framwork that offers wide support of annotations throughout the whole project. There, annotations provide a alternative way to specify deployment descriptor information.

Advantages of annotations are that configuration values are placed right next to the element in the source code it is related to. This helps to provide intuitively readable code. Another advantage is that development environments support the developer using in annotations. Since they are actual Java types, tools like *code completion* can be used.

As a disadvantage annotations are placed within the source code. This requires recompilation if the configuration has been change. If configurations are placed externally, e.g. in XML files, recompilation of the source code is not necessary.

In practice it's usually a pay-off between rapid developed readable code and purely declarative configuration in XML files. This pay-off has to be estimated by the individual developer.

[32] See e.g. [Mica] or [Ull07], p.1301.

Chapter 3

Enterprise Content Management

The term *Content* is the second major topic this thesis deals with. More specifically, this chapter elaborates on the characteristics of content and what content really is, how it is managed in enterprises and what special use-cases there are that further characterize content in the real world. Finally, the standard Java Content Repository will be introduced as it will later be used during the implementation of a framework that accesses content.

3.1 Terminology

In this section basic terms around the field of content are introduced and defined.

3.1.1 Fully structured data

Fully structured data has already been introduced in section 2.3.1.

3.1.2 Unstructured data

Unstructured data is data that has - from a given point of view - no recognizable or processible inner structure. A file system for example doesn't know about inner details of the files it is storing. A relational database doesn't have knowledge about binaries it is storing in a BLOB field. On the other hand, Microsoft Excel knows how to open and interpret the Excel document that is stored in the file system or in the database. Obviously, the kind of structuredness changes as the viewpoint changes.[1] [Abi97] is also referring to this as *raw* data.

3.1.3 Semi-structured data

Semi-structured data is often referred that is neither fully and strongly structured nor completely unstructured. Other definitions, like the one used in here, consult the same definition as used for fully-structured data but turn it around. One of the most frequently cited sources, [Abi97] gives the following definition. Its structure was already used for the definition of fully-structured data, given in section 2.3.1.[2]

Not pre-defined Unlike relational databases the structure of data doesn't have to exists at the time the data is being introduced. The structure is mostly implicit it will evolve with the introduction of new data.

[1] See [Abi97], p.1.
[2] See [Abi97].

Implicit Structure information is nested into the actual data. XML is a good example of semi-structured data. It mixes structural information (the *tags*) with content. That way, the structure is implicit, can be reproduced by reading the data and can be changed by added new data containing new structure information. Also, the responsibility of delivering the structure is moved to the data itself.

Irregular and incomplete The structure of a given set of data may vary in different instances, e.g. certain fields may be ommitted, other field may be added. This also means, that semantically identical things (e.g. an address record of a customer) may be structured differently, e.g. one time as a string, the other time as a tupel.

Indicative The structure guides (indicates) the way the data should look like. It doesn't serve as a strong constraint.

Often the terms *"'self-describing"'* and *"'schemaless"'* are used as synonyms for the term semi-structured.[3]. A popular way of modelling semi-structured data is to store them in a hierarchical form representing a labelled graph[4]. In this form, there is no distinction anymore between the schema and the data.[5]

Unstructured data usually occurrs in small amounts, e.g. single files of unstructured data. Mostly it is embedded into a set of semi-structured or fully structured data. Examples areas of usage for semi-structured and unstructured data are biological databases, database integration, multimedia content or the World Wide Web.[6]

Adding structure to a certain degree to formerly unstructured data, e.g. a plain written text, allows to significantly increase the quality of query results on that document. This is often achieved by using XML for enrichring text-based documents with markups information.

The fact, that structure and data aren't distinct but combined makes exchanging semi-structured documents very easily exchangeable. A popular example is XML, which contains the actual data embedded in tags that give information about the structure.

3.1.4 Content

It is hard to find clear and common definitions of what *Content* really is, neither in IT-related literature nor in dictionaries[7]. Commonly it is also reffered to as "'Media Data"'[8]. From this point and together with a look at the occurrences of the word throughout the world of software and internet, it can be assured that content is something that could be referred to as media data: videos, songs, free written text, recordings, spreadsheets, PDF documents and the like. In conjunction with the given definitions of the structuredness of data, it can further be refined as data, that includes, but is not restricted to, fully structured, semi-structured and unstructured data.

Sometimes also the term *Fixed Content* is used. While the structuredness of the data is still the primary characteristic, this should emphasize that the content is generated once and then immutable over its complete lifetime. This opposes dynamic content, that is generated on demand. Nonetheless, due to its nature nearly all the content belongs to the group of fixed content. Looking at videos, sounds, Office documents etc. it seems to be obvious that most of them aren't generated on demand.

[3] See e.g. [SA00], p.11.
[4] See [PB97].
[5] See [KA89], page 1 and 2.
[6] See [PB97].
[7] To be precise: the latter of course provides explanations. But they are not very helpful in this context.
[8] For a good reception of received opinion see here: *http://en.wikipedia.org/wiki/Content_%28media_and_publishing%29.*

3.1.5 Content Management / Content Management Systems

Content Management describes the management of larger quantities of content. *Content Management Systems* are specialised in managing non-fully-structured content. Often they are also specialised in managing the most popular and most likely occurring types of content, such as XML and HTML documents, Microsoft Office documents or PDF documents.

3.1.6 Enterprise Content Management

Enterprise Content Management, ECM, deals to a large degree with content management but furthermore attempts to embed content management into enterprise environments. Hence enterprise content management becomes part of organizational processes and the demand to meet business goals. The *Association for Information and Image Management*[9] gives the following definition which is also widely agreed upon as being somewhat offical:

> *"'Enterprise Content Management is the technologies used to Capture, Manage, Store, Preserve, and Deliver content and documents related to organizational processes."'*[10]

To some degree ECM is very similar to content management. In large enterprise and organizations though, many task involved in managing different content become significantly more difficult due to the mere size of the systems, to large quantities of documents and to a large number of users accessing this content. Also, for most organizations, big amount of content data stored on their systems can be seen as mission-critical information. Hence the priority is set reasonably high.

The amount of content-based data in organization is also rising tremendously in recent years. Due to technological shifts towards multimedia and internet much more information is being created and shared around the world. Organization also have much more access to data that is necessary to be stored might later be retrieved. The introduction already mentioned [RK05]. It states that about 80% of all information stored within organizations are content data.

3.2 Typical Use-Cases of Content

So far the term content has mostly been defined through its level of structuredness. In the context of this thesis it's important to discuss how content is handled by applications. This is especially of major interest as this handling differs from the handling of fully structured data in enterprise application. This section will present some of the important use-cases that occur in conjunction with content in real-world applications.

3.2.1 Full-text search

Fully structured data in enterprise application is mostly searched by performing comparison queries on single fields. A striking example is the syntax of SQL which bases on operations on single fields. A full-text search on the other hand means searching within multiple fields, e.g. in all fields of a table or, in terms of an object-oriented programming language, in all fields of an entity. This is also called *content-based query.*[11] With SQL this can only be archieved by

[9] Abbr.: *AIIM*. See *http://www.aiim.org*
[10] See *http://www.aiim.org/about-ecm.asp.*
[11] See [AH00].

combining queries on single fields. In content now, the focus lies much more on objects that contain multiple fields at once. This is often be referred to as a *document*, emphasizing that it is more than just a single field. Accordingly, searches are often spanning the whole document. If for example a certain string would be searched within a book store application, a match could be found either in the title, in the full-text of the document, in certain keywords or in the abstract. This is a popular use-case in content applications.

3.2.2 Discovery Querying

As explained in section 3.1.3, semi-structured and unstructured data don't have a pre-defined structured that is well known a priori. Hence, queries on content often don't follow any predicate logic at all. Rather users are *discovering*[12] the structure and its containing content step by step, e.g. by querying into the hierarchy of content and querying again based on the actual data found at the respective position. This appears obvious looking at the fact that the structure and occurring fields can change anywhere within the structure. This is a use-case that is mostly performed by humans and hence has implications on the way the structure is created. It is often supposed to be intuitive and human-readable, e.g. by using well-sounding names for labels within the graph instead of kryptic abbreviations or the like. Content application often provide a way to support in creating those structures that make discovery querying easier.

3.2.3 Searching Binaries

Often searches should not only be performed over text data, as possible with SQL for example, but also over binary files like PDF documents or Excel worksheets. To be able to do this, there must be knowledge about the particular file format and about its inner structure in order to extract text. Relational databases do not support this kind of query per se. Content management application often have built-in support for searching binary documents. Usually meta information is used to determine the concrete file format which is then openened and extracted by special components.

3.2.4 Inexact Querying

In relational databases inexact querying is poorly supported. SQL supports the % operator to search for the occurence of a string within a text field. Content is rarely exactly specified information but rather free written text from various sources, data in different languages or the like. Expected matches are often not only exact strings but rather matches or nearly-matches on a more semantical level. There are three popular types of inexact queries that occur in conjunction with content.

- **Fuzzy Search** This kind of search finds strings that are similar to the original string. The *similarity* depends on the respective use-case. The most popular kind of similarity is the phonetic similarity that finds words, that sound similar to the one originally searched for. Another example is a search that corrects common spelling mistakes.
- **Synonimical Search** This kind of search finds words, that have the same or similar meaning than the one original searched for. This makes especially sense in querying large numbers of documents that are human-written. The similarity in this case is of a semantical nature.

[12] See [PB97], p.14.

- **Cross-language Search** This kind of search finds matches of words, that have the same or a similar meaning in other languages. This is of particular advantages when dealing with multi-lingual documents. In internationally created knowledge bases for example a certain article could have been created in English but not in German. A query for a German keyword should find the English article though. Without Cross-language search, no article at all would be found.

3.2.5 Versioning / Merging

Versioning is a popular use-case in enterprise applications, too. But unlike with content, where individual fields are under version control, enterprise applications tend to create another scope for versioning, an entity-wide scope. That means that whole business entity objects are under version control.

Another difference to content is the usage of merging in enterprise systems. In much cases merging doesn't make sense for enterprise data. The name of a customer for example should never be merged from two different changes into a new result. As content management deals with large textual documents, merging is often necessary in order to allow efficient, concurrent working on documents.

3.2.6 Internationalization

Internationalization is the possibility to store documents in different languages and to retrieve it for a certain language. Other features are imaginable to be combined with this, the e.g. cross-language search introduced earlier.

3.2.7 Summary

Not only does content have different technical attributes, the structuredness, compared to data in enterprise applications. Especially the handling of the data makes a major difference. Furthermore these use-cases aren't supported by relational databases which are the most often used databases in enterprise applications. This makes clear, that applications of these use-cases in enterprise application is not trivial.

3.3 Market Overview

3.3.1 Product Categories

Unlike the market of enterprise application frameworks the market for content management solutions is much more diverse. It could roughly be separated into the following categories:

- **Custom applications** that are specialized in addressing some special content use-cases.
- **Content Management Systems** They mainly evolved out of content-centric internet applications. These products are characterized by providing specialiced maintenance user interfaces (eg. Rich Text editors for HTML content), the ability of generating output in different formats and the integration of simple implementation logic e.g. to cover publishing workflows or the like.
- **Enterprise Content Management Systems** As mentioned about in section 3.1.6 these system cover complex content lifecycles and are integrated in complex enterprise

environments. Example products of this category are EMC Documentum, Microsoft Sharepoint or Day Communique.[13]

3.3.2 Java Content Repository

Java Content Repository (JCR) is a standard developed under the Java Community Process[14] as JSR-170. Its first version was finalized in 2005. It was the first official standard to be passed dealing with content access. It allows application developers to use a standardized API that hides from the underlying storage implementing the standard as well as of all technical details behind it. As JCR has been chosen to be used in this thesis, it will be presented in further detail later in section 3.4.

3.4 Java Content Repository

This chapter introduces Java Content Repository, a standard which was elaborated, defined and released under the Java Specification Request as JSR-170[15]. At its core, it specifies an abstract repository model inteded to store content data as well as a Java API as the only means to access the underlying repository. JCR is widely seen as the first successful attempt to introduce a standard API for accessing content.

3.4.1 History

In 2002 an expert group led by David Nuescheler began to work on JSR-170, the first version of what later became popular as Java Content Repository. That expert group consisted of 22 companies and organizations, e.g. Day Software, Apache Software Foundation, IBM, SAP, Hewlett-Packard, BEA, Sun and Software AG. It shows the recognition of this standard throughout the industry of companies dealing more or less with content storage.

JSR-170

On 17th of June 2005 the final specification of JSR-170 was released carrying the version number 1.0. It mainly contains a specification document and a set of Java APIs. The reference implementation has been moved to Apache Software Foundation under the name "'Jackrabbit"'[16] mainly to provide a living open-source implemementation that can easily be adopted. Apache Jackrabbit is still the most popular open-source implementation of JSR-170.

JSR-283

In fall 2005 the work on JSR-283 began, popularly known as "'Content Repository For Java Technology API 2.0"' or short "'JCR 2.0"'[17]. The early review phase was closed in October 2006, the standard is currently planned to be released in January 2008. So by the time of writing JCR 2.0 is only available in a non-final version with no products providing support yet. For this reason JCR 2.0 will not be taken into account in this thesis. The basic principles however will be the same and improvements or introduction of new features will, although helpful in practice, only have a minor impact on work and statements of this thesis.

[13] Links can be found in appendix A.
[14] See *http://www.jcp.org*.
[15] The specification can be found at [(JC05a].
[16] See section 3.4.9 for a detailed introduction on Apache Jackrabbit.
[17] The specification process can publicly be followed at [(JC05b].

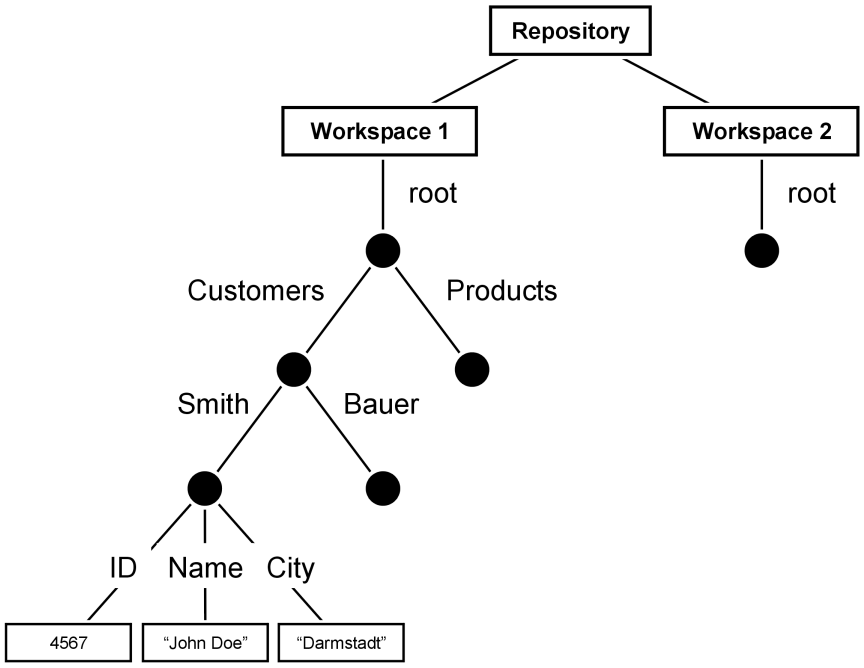

Figure 3.1: Hierarchical Repository Model of JCR

3.4.2 JCR Concepts

Roy T. Fielding summarizes in [Fie05] that JCR is a *"'generic application data super store [that] is expected to manipulate and store structured and unstructured content, binary and text formats, metadata and relationships"'*. It further provides integrated content services such as *"'uniform access control, locking, transactions, versioning, observation, and search"'*[18].

JCR defines a set of Java interfaces and is intended to be implemented by various products. Due to the assumption that different use-cases require a different set of functions made on the repository, a system of levels was introduced to describe which set of functions a certain product offers. E.g. a news ticker system accessing the news it displays from a JCR-compliant repository only needs reading access. On the other hand a content managament back-end application will most likely need read and write access to allow users to add, edit end delete content. The concrete system is defined as follows:

1. **Level 1:** The implementing product offers read access like browsing the content structure, reading the values of stored content and export the content. In addition it allows to search the repository using XPATH.

2. **Level 2:** In addition to all Level 1 functionalities it also offers write access, like adding and deleting content and maintaining the actual content. In addition, custom node types can be defined and used. Level 1 is a complete subset of Level 2.

3. **Optional features:** An additional features was specified and may be implemented independently:

   - Transactions

[18] Taken from [Fie05], page 4.

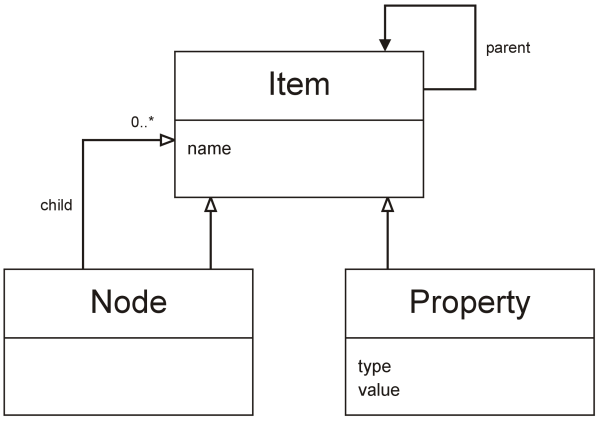

Figure 3.2: UML diagram of Item, Node and Property of the JCR Repository Model

- Versioning
- Observation
- Locking
- SQL

3.4.3 Repository Model

The basic concept in JCR is the hierarchical representation of data, similar to XML. Starting with a single mandatory root item, every item has a name and may have child items.[19] Items themselves are seen as abstract and thus are either nodes or properties, both inherited from item. Nodes can have further child items whereas properties cannot. Properties have a certain type and a value. The types are defined in the specification and contain most of the primitive types known from programming languages, such as String, Integer or Stream.

In other words, nodes are used to structure the stored data, properties are used to store the actual data themselves. Visually these items are building a tree where nodes are the branches and properties are the leaves. Every item can be absolutely addressed by going down the tree starting at the root item and adding up the names of the visited items to form the address. As the slash symbol is used as delimiter between the single names, the resulting address looks pretty much like a URL. One of the properties in figure 3.1 can for example be addressed by */Customers/Smith/City*. In practice, an important task in designing JCR-based applications is the definition of a structure that leads to speaking addresses and thus intuitive trees.[20]

The following itemization lists the most important terms used by the JCR specification:[21]

Item, node, property The elements forming the hierarchical tree of data are called items by being further specified by one of the types: node or property.

Repository A repository represents one instance of a content repository, e.g. on a server. It can contain several workspaces. It can be compared with one installation of a database server.

[19] See figure 3.2.
[20] This has already be mentioned in section 3.2.2.
[21] See also figure 3.1.

Workspace A *Workspace* has one mandatory root node which then can contain a tree of items. Workspaces can separate different kinds of content on the highest level. It can roughly be compared with databases within a database server.

Session Unlike the terms repository and workspace, which are used on a conceptual level, the term session is rather technical. Sessions can be seen as handles onto workspaces. When working with the JCR API, a session handle has to be retrieved from the repository. Every client accessing the repository has to obtain a session first.

3.4.4 Node Types

Node types allow to define and contraint items in a similar way a database schema defines the structure of databases[22], tables and fields[23]. Basically, every node must be assigned one so-called primary node type on creation. This assignment is immutable over the life time of a node. Within that node type definition, it is predefined which properties and which child nodes the node has. Unlike the very explicit and static schemas used in the relational model[24], JCR allows more flexibility.

Two different types of nodes types exist in the terminology of JCR: *primary node types* and *mixin node types*. Every node has exactly one primary node type assigned. Node types can be inherited for definitions of new node types, while every node type directly or indirectly inherits from *nt:base*. This is comparable with inheritance hierarchies in Java, where every object is directly or indirectly derived from *Object*. Additionally none or more mixin node types can be assigned. That way, the basic behavior of a node is being described by its primary node type while eventually existing mixin node types can add additional behaviour. Mixin node types can be compared with interfaces in Java. Every class inherits from exactly one super class, but can implement none or multiple interfaces.

The JCR Specification defines a set of built-in node types, primary node types as well as mixin node types. These are intended to provide a basic functionality out-of-the-box and are also used by the specification itself to solve fulfill technical requirements. For example some of the built-in primary node types are *nt:folder*, *nt:file* or *nt:resource*. They are supposed to be used whenever content should be stored that is semantically close to folders, files or resources in the common sense. The built-in mixin node type *mix:versionable* is used by the specification to mark an item as being versionable. This example also outlines that primary node types are widely used to define semantic characteristics whereas mixin node types can add rather technical characteristics.

Looking again the built-in primary node types, it first looks like they are intended to minimize the needs to define custom node types for the most popular use-cases[25]. But there is a big advantage coming along: as JCR defines an open standard tools will evolve (and in fact have been evolved already) that can handle content repository generically, like the well-known Windows Explorer can be used to browse file systems. But these tools may be able to handle nodes of built-in node types in a specific (and hence in a better) way, increasing their usability. In practice, this already happens with a lot of tools making it advisable and best-practice to involve built-in node types as much as possible in customized content models.

[22] More generally, the term "'schema"' is widely used to define a certain domain specific data model, e.g. for for XML or even in object-oriented programming. In these cases, the term "'logical schema"' is also used.

[23] A very spongy comparison could also be made to sets of meta-data associated with certain file types, e.g. image files may carry meta-data like width, height and resolution.

[24] Compare the definitions of fully structured and semi-structured data, specifically indicative vs. constraining structures. See sections 2.3.1 and 3.1.3.

[25] E.g. a storage system for files could easily be created by just using the build-in node types nt:folder and nt:file.

It has to be noted, that JCR also allows some operations at runtime. E.g. mixing node types can be assigned on a per-case basis during runtime which also adds some flexibility to the content model.

3.4.5 JCR API

The base package of the JCR API is *javax.jcr.**. It defines classes and interfaces to connect to a repository, to access the content strucure, to add, edit and delete content, to query the repository and to use the optional features (if existant for the actual implementing product).

All access to the repository is done using this Java API. As the underlying storage is completely hidden by this API there is no way around, e.g. to access the database directly (in fact, it is even unknown whether a database is the underlying storage, it could also be the file system or a connector to another system). On the upside, applications written on-top of this API don't have to care about storage details and can be used with every implementation [26]. Also, driver issues, connection establishing and the like is also hidden. These became administration tasks in JCR, done by the administrator of the repository. The only necessary remainder is code to lookup the repository, which in JEE application can be simplified by using JNDI or dependency injection.

3.4.6 Querying

The default query language in JCR in XPath[27]. XPath was originally created by the W3C[28] and intended to access parts of XML documents but due to the similarities between the hierarchical repository model of JCR and hierarchical structure of XML XPath was chosen as the query language of choice. As an optional feature, JCR specifies a customized dialect of SQL[29].

3.4.7 Versioning

Versioning allows to restore earlier states of data stored in the repository. Versioning can be enabled for certain nodes (at design time or runtime) by assigning the mixin node type *mix:versionable*. The version feature in JCR was designed according to the specification made by JSR-143 about "'Workspace and Configuration Management"'[30] Versioning in Jackrabbit is implemented to show behaviour similar to typical versioning systems like CVS and Subversion. The JCR specification also covers a mechanismn to merge concurrently edited content.

3.4.8 Transactions

JCR specifies transaction support for repositories according to the Java Transaction API[31] This allows to integrate JCR into environments and use-cases where transactions are necessary to ensure data consistency. Therefore JCR also specifies both container-managed and bean-managed transactions. See also section 2.3.5 about transactions.

[26] Here, the compliance level has to be taken into account. An application written for a Level-2-compliant repository, in other words a system that does write operations on the repository, can not be used with a Level-1-compliant repository. Although the API is the same the Level-2-related operations of the API don't invoke any actions.

[27] The specification of the current version 2.0 can be found here under *http://www.w3.org/TR/xpath20/*.

[28] The *World Wide Web Consortium*. Its homepage can be found under *http://www.w3.org*.

[29] See also section 2.3.4 about SQL in relational databases.

[30] See [(JC01].

[31] See [Micb].

3.4.9 Apache Jackrabbit

Apache Jackrabbit is the official reference implementation of JCR and remains until today the only fully featured available open-source implementation of notable maturity. By the time of writing the newest version carries the number 1.4. It implements the full specification of JSR-170 while preparations for the upcoming version JSR-283 are already under way. Apache Jackrabbit reached an decent community as gained a notable recognition amongst tools vendors throughout the industry. Several case studies approve that Jackrabbit is used in large-scale production projects all over the world.

3.4.10 The mapping framework Jackrabbit-OCM

Jackrabbit OCM[32] is a sub-project of Jackrabbit that has become part of the official release of Jackrabbit with version 1.4. It provides a framework that allows to map content accessed via JCR into Java objects. Besides simple mappings of JCR properties into Java attributes it also allows more complex mappings like inheritance or polymorphism. A component called *Object Content Manager* can connect to content repositories through the JCR API and perform persistence operations on content. This way, OCM allows to persist Java object into JCR-compliant repositories and hides from the details of the JCR API. Due to its capabilities to map content into Java objects, Jackrabbit OCM will be used later in chapter 5.

[32] More information about Jackrabbit OCM as well as further details about how to set it up and use it can be found on the project homepage under *http://jackrabbit.apache.org/ocm*.

Chapter 4

Implementation Models and Decision Process

So far enterprise applications and content applications as well as their different characteristics have been introduced. This chapter will now make use of this knowledge in order to describe implications on design decisions in software development projects.

The goal of this chapter is to introduce different implementation models that can be applied for given project situations. A proposed decision process will support architects and application developers with guidelines and evaluation methods to determine a suitable implementation model.

4.1 Problem Setting

The problem setting as well as the whole topic of this thesis is coarsely located in the field of modern business applications. Although the information given in this chapter claim to have a wide scope of validity in terms of different application frameworks, platforms, technologies, programming languages and the like, it should be clear, that the thesis has been developed in the field of Java EE 5 and JCR. The information should be taken with care with rising technology distance, e.g. when trying to apply the principles to mainframe applications or embedded real-time systems.

Figure 4.1 shows an business entity object *Customer* as it would be identified in an early project phase as part of a larger entity model. This object contains certain attributes like an *ID*, a *name* and an *address*. At that stage of development progress all entities are still independent of technology-specific details.

In a further step, e.g. the next project phase, some technology have to be made in order to be able to design the architecture and start implementing the application. To support this decision will be the essential outcome of the current chapter.

Please note, that it's not necessary to really be in that early project phase. The author just wants to emphasize that the information given in this chapter will have fundamental influence in technological decisions. This is much less obvious when talking about existing projects. Other scenarios like refactoring situation would be suitable as well.

4.2 Implementation Model Decision Process

The decision process developed during this thesis is intended to support architects and application developers in determining a proper implementation model or architectural strategy

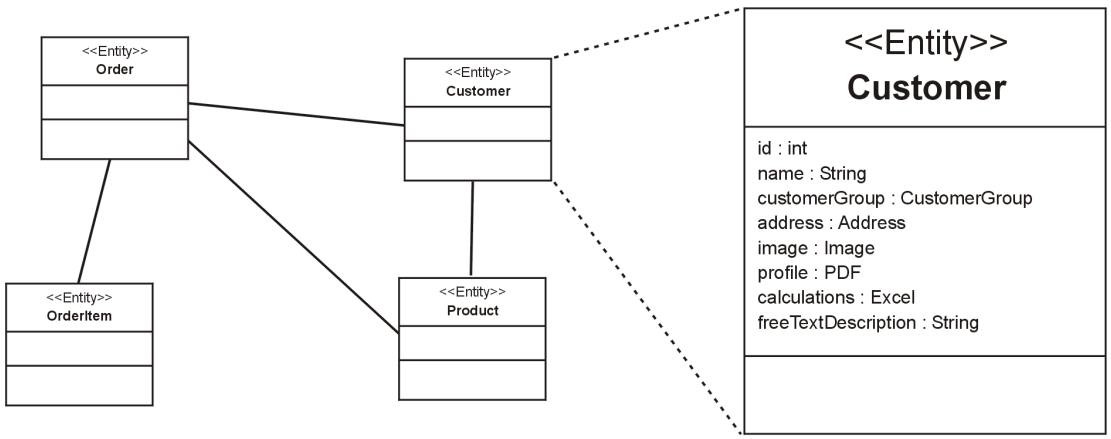

Figure 4.1: UML Analysis Model of Business Domain and Entity Object 'Customer'

respectively. A rough overview of this process is illustrated in figure 4.2. Certain factors are influencing the decision. Different implementation models are the outcome of the decision. Further refinements of this process will be presented during this chapter.

4.2.1 Related Work

In the context of software engineering two methods exists that are similar to the decision process introduced in here. The first one is the ATAM, the *Architecture Tradeoff Analysis Method*[1]. It has been developed by the Software Engineering Institute (SEI) of Carnegie Mellon University[2] and is intended to provide *"'software architects a framework to reason about the technical tradeoffs faced while designing or maintaining a software system"'*[3]. On top of ATAM the SEI has built another model in order to quantify architectural decisions from a rather economic point of view. This is CBAM, the *Cost Benefit Analysis Method*[4], and it *"'causes the stakeholders to quantify the benefits as well as the costs, dependencies, and schedule implications of architectural decisions"'*[5].

The process elaborated on in this chapter can be seen as an extension to both models. Especially the methods introduced in section 4.4 are similar to the methods proposed in the context of CBAM. Also the basic intention of both the process of this thesis and of CBAM, are similar: the attempt to estimate total costs as well as benefits of all possible options in order to be able to compare the total sum of benefits minus costs. This is intended to lead to the best option for a given situation.

4.2.2 Influencing Factors

Several different factors would have to be taken into account in order to determine the right implementation model for a given problem setting. This thesis will not investigate all of them. Nonetheless here is an incomplete and exemplary list:

[1] See [Unia].
[2] See *http://www.sei.cmu.edu*.
[3] See [RK01]. Furthermore see [JA] for more information about ATAM.
[4] See [Unib].
[5] See [JA01].

- Formal functional and non-functional requirements as officially collected during analysis.
- Existing knowledge within the development team.
- Existing IT landscape.
- Political decisions, e.g. which vendor to prefer over another.
- Favours and gut feelings of architects and team leads.

However, only a small number of factors will be incorporated into the decision process. These are factors that are directly related to enterprise applications, content applications and to the data processed in these applications. The concrete factors will be explained in section 4.3.

4.2.3 Implementation Models

Theoretically there are lots of possible implementation models to choose from and even more architectural decisions to make for a given problem setting. This thesis focusses on the following options which are also part of the illustration in figure 4.2:

Enterprise Application The outcome of the decision process is that the application will be implemented as an enterprise application as outlined in chapter 2.

Content Application The application will be implemented as a content application according to descriptions in chapter 3.

Hybrid Implementation Both, an enterprise application and a content application will be used for implementation. They are co-existing besides each other.

The first two implementation models will also uniformly be referred to as **single-system implementations**. In the following section the differences between enterprise and content applications will be presented in order to find factors that are influencing the decision process.

4.3 Enterprise Applications and Content Applications

Chapters 2 and 3 introduced the basic theory of enterprise applications and content applications. If the outcome only consists of these two combinations then it is necessary to find the factors that decide over one or another of them. For that reason a comparison will be done in order to find the clear points of distinction between both kinds of applications. These points will become the individual influencing factors in the decision process model. Other characteristics can then be ignored.

4.3.1 Differences between both kinds of application

Three factors have been identified that clearly distinguish both kinds of applications:

- The structuredness of data that the systems mostly work with.[6]
- The characteristics of the handling of that data. This means the way that data is created, structured, organized, retrieved and queried.[7]

[6] See section 2.3.1 about structuredness of data in enterprise applications and section 3.1 about structuredness of content data.

[7] See sections 2.3 and 3.2.

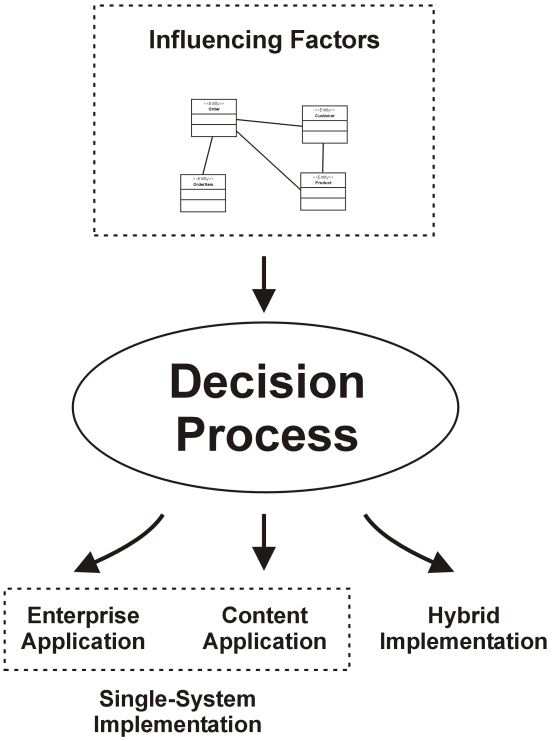

Figure 4.2: Decision Process - Influencing Factors, Implementation Model as Outcome

- The implementation logic that the systems need to realize in order to work properly.[8]

These three factors are the only factors that clearly distinguish both kinds of data. Table 4.1 lists them and their individual values for both kinds of systems. In order to classify a system to either one of the two kinds it is necessary to determine their values according to the three factors.

	Enterprise Applications	**Content Applications**
Structuredness of data	Fully-structured	Fully structured, semi-structured, unstructured
Implementational logic	Complex, business-oriented logic implemented with programming languages	Specialised content features, no complex business logic, data-centric
Handling of data	Fully-structured querying paradigms	"'content use-cases"', full-text search, versioning, internationalization

Table 4.1: Distinction between Enterprise Applications and Content Applications

4.3.2 Implications on Design Decisions

Three factors have been identified that clearly distinguish enterprise applications and content applications. Consequently, these three factors are of major interest when deciding which implementational model to choose as they are the only remaining variables. All other factors

[8]See section 2.2 about business logic in enterprise applications, section 3.1 and 3.2.

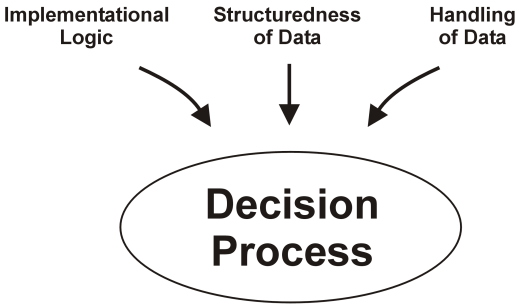

Figure 4.3: Decision Process Extended by Influencing Factors

are then independent of the choice of technology. Figure 4.3 illustrates a modified input layer of the decision process.

The implementational logic that is required to be implemented in the target application differs from the other two factors. It is a rather external requirement that explicitly exists after requirements engineering has been done for the application. Roughly speaking, either a complex business-oriented logic as introduced in section 2.2 has to be implemented or not. Hence this factor doesn't have to be examined any further, it is seen as an outcome of the project's requirements analysis.

The two other factors need further examination in order to find out what role they are playing in the decision process. They are both of major interest in this thesis as they are mostly characterized by data. They can also be called *data-driven* factors.

4.4 Qualitative and quantitative methods

The previous section determined that data-driven factors are playing a major role in the decision process. It is necessary to characterize the data and the target application according to table 4.1. This section will propose three methods that should guide responsible architects in a concrete way to find the right answer necessary for the decision process. First, a terminology about different kinds of data is introduced:

- **Enterprise Data** Fully-structured data that conforms to the characteristics of enterprise applications as explained in chapter 2.
- **Content Data** Mostly semi-structured or unstructured data that conforms with the characteristics introduced in chapter 3. Especially the special use-cases that might be applied for content data is of major interest.
- **Data Type** This term is used to generically express which kind of data a particular attribute is of.[9]

The important task within the decision process is to determine to which kind of data the attributes given in the entity objects are belonging to.[10] Usually different kinds of data types exist within the same entities. First it has to be figured out how the relations between

[9] It should not be mixed up with the traditional meaning in programming. It's a more semantical classification of data used for the purposes of this thesis.

[10] Again, this is not restricted to projects that are in early stages, it also applies for existing application.

Survey Method - Binary Values - Entity 'Customer'					
	Full-text Search	Binary Search	Versioning	I18N	
id	0	0	0	0	
name	0	0	0	0	
customerGroup	0	0	0	0	
address	0	0	0	0	
image	0	0	1	0	
profile	1	1	1	1	
calculations	1	1	1	0	
freeTextDescription	1	0	1	1	
Total	3	2	4	2	**11**
% of all attributes	38%	25%	50%	25%	

Table 4.2: Example result of the Survey Method using binary values

both data types is at project level because technological decision have to be made at project level, too. In case the mixed implementation will be chosen, the particular data type of every attribute becomes important. In the following three different methods are proposed to achieve these results. They are able to achieve both results as follows:

1. To get an overall result for the whole system in order to be able to decide which model is suitable.

2. In case the mixed model will be chosen, then these methods also deliver results for individual attributes. They help to decide where the attributes will be implemented.

4.4.1 Survey Method

This method rests on explicit valuation of every single attribute existing throughout the application. In section 3.2 use-cases have been introduced that are necessary to be done with content data but aren't possible to achieve with standard persistence mechanisms of enterprise applications. On the other hand, this also means that it is might be required to perform those use-cases upon a certain amount of the data.

Those use-cases can now be taken as a checklist that has to be applied for each data attribute. It can be imagined as answering the question *"'How desirable would it be to be able to perform this use-case for this attribute?"'*.

There are different ways to perform this survey, each requiring different amounts of effort, providing a different level of granularity and each leading to slightly different results. The recommended ways to perform it are explained in the following:

- A sheet has to be created like the one in table 4.2. Only binary values like *0* and *1* or *yes* and *no* respectively are used. The numbers for every attribute are added up to a *individual total sum*. The result would be how many content use-cases would be required for this particular attribute. Based on that the attribute could be classified as either *enterprise data*, when having a low individual total, or as *content data*, when having a high number. A general problem here is, that it's difficult to tell an exact limit. This method assumes that every content use-case has the same importance as well as the same level of desire to the customer as it is impossible to assign different weights. A sample result is illustrated in table 4.2.

Survey Method - Weighted Values - Entity 'Customer'						
	X	Full-text Search	Binary Search	Versioning	I18N	
X		2	3	2	5	
id	1	0	0	0	0	
name	1	0	0	0	0	
customerGroup	1	0	0	0	0	
address	1	0	0	0	0	
image	1	0	0	1	0	
profile	5	1	1	1	1	
calculations	3	1	1	1	0	
freeTextDescription	2	1	0	1	1	
Total		18	24	22	30	**94**

Table 4.3: Example result of the Survey Method using weighted values

- In case either use-cases or attributes or both should be weighted differently, numbers in a certain range, e.g. from 0 to 5, can be used. That way either the importance of certain features or the individual factor of certain attributes could be taken into account. A sample result can be found in table 4.3.
- Another kind of weighting could be done on entity level. That way the importance of certain entities over others can be taken into account. This is especially important in estimating the content factor for the whole system. A central and important entity might likely be to influence the total result more than unimportant, minor entities.

This method is intended to be tailored for individual requirements on a per-project basis. Furthermore, throughout this thesis not enough results could be collected to give further recommendations or experiences about concrete numbers. Generally speaking, the more complex the method is, the better the quality of the results can be but also the more complex it will be to perform the survey.

The survey method delivers results on different levels: on data level, on entity level and on system level:

- Data-level results show which attributes should are classified as enterprise data or as content data and hence lead the decision process which attribute is modelled into which of the two systems.
- Entity-level results decide whether a certain entity contain a significant amount of content or not.
- System-level results help to estimate which approach is the right one. A very low amount of content data leads to a realization of the system as an enterprise application, a very high amount leads to a content application. Numbers somewhere in the middle lead to a mixed approach.

Finally it is necessary to make decisions based on concrete numbers or percentages. For example there could be a rule like *"'If the survey method delivers a content ratio of less than 25, the system would be realized as an enterprise application. Above 75, it would be realized as a content application. In between the mixed approach would be done."'* Those numbers are hard to tell and depend very much on the concrete project situation. Also, during this thesis couldn't be enough experience collected to be able to give those numbers.

4.4.2 Data Type Method

This method takes the data type into account. The following test shall be performed for every attribute:

1. If the data type is *binary*[11] then this attribute is most likely content. A high number has to be assigned, e.g. *10*.
2. If the data type is Text then it might be content. This is because in practice content often appears to be unstructured or semi-structured, but text-based. Examples are all kinds of free text on web pages, newspaper, free text descriptions and the like. A medium number has to be assigned, e.g. *4*.
3. Other data types are unlikely to be content. A *0* has to be assigned.

Attention should be paid to the difference of data types on different levels. Best suited for this test are analysis diagrams as they usually contain the data type from a pure business point of view. In design diagrams and implementations the data type is often replaced due to technical constraints, patterns, performance issues or something like that. Finally, it is not recommended to use data types from representations in databases. On this level data types often get converted. For example the *Customer* entity has an association to a *Customer-Group*. In analysis as well as in design and implementation this is always recognizable as an association. In the database, this most likely gets converted, e.g. to a String representing a foreign key. So higher levels are more recommended to be used for this test.

The downside of this method is that the results delivered don't allow precise estimations. But then again this doesn't require human actions, it can very easily be automated (e.g. by a Eclipse plug-in or Ant task). Also, the information available in early-staged analysis diagrams (like the example *Customer* diagram given in this chapter) is sufficient to perform this test because only the data type is required to perform this test. Hence it can be of great value to support decisions of large projects in early project phases. It is imaginable that this test can continuously deliver a current estimation of the whole system.

4.4.3 Semantic Method

This method is similar to the Data Method but aims for existing projects rather than projects in early stages. It is also possible to automate the execution of this method. If applied for *existing projects* it can deliver valuable information supporting refactoring decisions.

The basic idea of this method is to find data that is only stored and displayed but not used within the business logic. Looking at a great number of content data it is outstanding that content data are very likely only used to be displayed e.g. in backend systems for maintenance purposes or on websites for presentation purposes.

Tracking down to a level that is analyzable, this fact can be described as the *non-occurrence of content-data within the business logic*. An implementation of this method has to follow each and every attribute and check whether business logic code make usage of this attribute. Logically a very good approximation should be possible.

The obvious downside of this method is that it is very hard to implement. Some of the problems are:

[11] This of course has to be substituted for the concrete model or programming language, e.g. replace it with *byte[]* or *InputStream* when working with Java or to the right data type when working on a platform-independent level.

- Due to programming techniques like polymorphism it might be hard to reliably identify each attribute within the source code.
- Depending on the architectural attributes might be transformed into other objects for some time, e.g. into data transfer object in order to be transferred between component or layers. That makes it hard to track.
- Modularization makes it very hard to track dependencies and occurrences. Recent trends to foster modularization and de-coupling worsen this problem.

It can be said that this third method is rather theoretical than a pragmatic approach that is ready to be used. Nonetheless, it provides further information about the nature of content data that might be useful and important at some point.

4.4.4 Implications and Summary

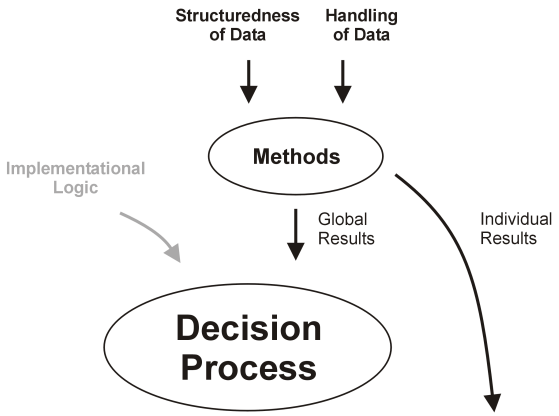

Figure 4.4: Methods deliver results to support the decision process

The three methods introduced in this section lead to qualitative and quantitative results about the data required to be processed within the system. Based on those results decision about the right model of implementation can be made. They further lead to documented results about the decision process and avoid decisions that are made just upon experience of architects.[12]

But the results not only lead to global architectural decisions. The next section will show that in case of hybrid implementations it has to be decided into which a particular attribute should be modelled. The results of the methods can also help with that decision. In summary, the two different kinds of results are delivered. They are also illustrated in figure 4.4.

- **Individual Results** Individual results are results per data attribute that are being documented while performing the analysis led by the methods. They indicate whether a particular attribute is either an *enterprise date* or a *content date*. They are of value if later during the decision process has to be decided into which system this date has to be modelled.

[12]This does not imply that it can replace experienced architects. It should rather prove decisions that were otherwise made of pure intuition and probably without being documented.

- **Global Results** Global results aim at supporting the architectural decision, i.e. they are influencing the implementation models as outcome of the core decision process. As individual results delivering the type of data for each and every date individually, the global result can be seen as the total sum of the individual results. This leads to global assumptions e.g. on how much of all data belongs to the category of enterprise data and how much belongs to content data.

4.5 Results of the decision process

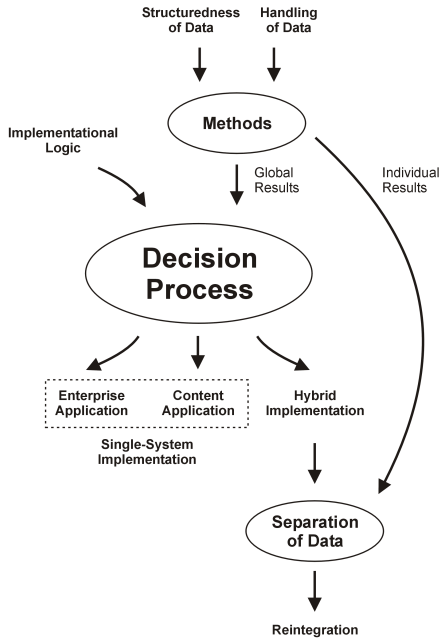

Figure 4.5: The complete decision process model

Figure 4.5 shows the complete decision process. Three influencing factors supported by the proposed methods support the decision process in order to find the proper model of implementation. If a hybrid implementation model is chosen, the methods also support the required separation of data. This section gives conclusions about the implementation models and shows that integration of data becomes a problem if a hybrid model has been chosen. This thesis will focus on cases with heterogeneous data and especially on hybrid implementations. Single-system enterprise application will be taken into account as a reference point, e.g. for costs behaviour which will be introduced in section 4.6.3. Besides that, single-system implementation in general will be ignored as tasks, problems and challenges involved in building single-system application are out of focus. They are also widely covered by available literature.

4.6 Hybrid implementations

In hybrid models, enterprise application and content application are running concurrently. First, the data have to be separated and modelled into either of the two target systems. This

can be determined by the individual results of the methods proposed in section 4.4. Chapter 5 describes the implementation of a hybrid system including a way to seperate the data.

4.6.1 Reintegration

A major issue when choosing the hybrid implementation model is the integration of both systems. Data that has, from a business point of view, been identified to belong to the very same business entity is now separated and might be located within two physically separate storages. The actual need for integration depends on the requirements. Different cases are possible, ranging from no necessary integration up to high integration with a lot of integration points. The degree of required integration is called the *Integration Complexity*.

4.6.2 Integration Complexity

The *Integration Complexity* denotes the overall complexity of integration that has to be done and is influenced by the following:

- The number of single integration points. The more integration points the higher is the integration complexity.
- The difficulty and maintainability of the implementation of certain integration tasks. This is the initial and ongoing effort that has to be put into the integration. The more effort that has to be put into integration over the complete software lifetime, the higher is the integration complexity.
- The degree of reusability of certain pieces of integration code for other integration points. The more single points of integration are reusable for other integration points, the lower is the integration complexity.

The higher the integration complexity is the higher the resulting costs will be. Specific costs are highly depending on the particular project but there is a tendency that resulting costs are rising directly proportional to the integration complexity. This problem only belongs to hybrid implementation models.

4.6.3 Cost development in hybrid implementations

The rationale of choosing a hybrid implementation is that required content features can basically be used out of the box. If choosing an enterprise application only, these content features would have to be implemented. This leads to two conclusions about cost development. They are also illustrated in figure 4.6.

- Using an enterprise application, the content-related cost drivers are mainly the number and the difficulty of the implementation of content features. This means that the content-uses introduced in section 3.2 have to be implemented in the enterprise application. In the author's experience these costs are traditionally high. Nevertheless, as the approach is straight-forward starting with a standardized enterprise applications, there is no extra initial effort that has to be put in up-front. In the diagram this can be recognized as the line starting from the origin without any intercept on the y-axis.
- Using a hybrid implementation model, the content-related cost drivers are mainly the integration complexity. This is the effort that has to be put into the system to bring data together that originally belonged together. This has been elaborated in section 4.1.

In the author's experience these costs tend to be lower than the costs of the previous point. But significant initial effort has to be put into setting up the hybrid system. This can be recognized in the diagram as the line having a rather big intercept on the y-axis.

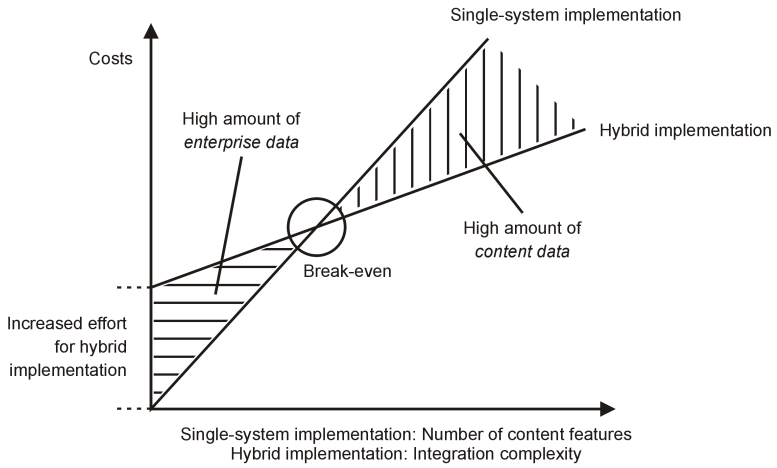

Figure 4.6: Costs development in relation to content features and integration complexity

Depending on the amount of content-data within the application and depending on the integration complexity, different decision are preferable. The diagram shows, that with low amounts of content data, it is recommended to use an enterprise application and add certain features by hand. At a certain point, the costs necessary for that even out with the initial costs necessary to set up a hybrid system. This is the reason why the Survey Methods tries to estimate the number of content features as well as weighted total numbers for content features. E.g. features, that are known to be costly to implement in enterprise applications, could get a very high weight in the survey method analysis.

4.6.4 Types of Initial Costs

Initial are now further be specified in order to eventually reduce individual cost factors.

- **Content application environment** Hybrid implementation models require a second environment to be set up. This means to actually introduce a content management application including steps like evalution, aquiring know-how and installation on test and production systems. Furthermore administration and maintainence, e.g. taking care of upgrades and the like, is required over the whole lifetime. Depending on the product, licence fees and maintenance costs can also be significant. This group of costs is not likely to be reduced.
- **Developer Know-How** Developer know-how and experiences must be available for each technological field that is being worked on. While know-how in enterprise applications might still be widely available, know-how in content applications and content integration is rare.
- **Time-to-market** This is the time required to deliver a solution. It also includes the time necessary to create prototypes and the like.

4.6.5 Maintenance Costs

Another type of costs ocurring in software development is the costs required for maintenance over the whole application lifecycle. Due to various estimations about 50%[13] or more of the total development costs are required for maintenance after the initial version of the software was finished. Re-integration of data as well as the implementation of content features in single-system enterprise application can generally be seen as an increase in complexity or, alternatively, as an increase of initial development effort.

[BDKZ93] states that the complexity of a software systems directly influences the expected maintenance costs. Barry Boehm developed in [Boe81] a basic equation to estimate yearly maintenance costs, which is

$$YearlyMaintenanceEffort = MaintenanceFactor(InitialEffort * AnnualChanges)$$

This formula shows the direct influence of initial development effort on expected maintenance costs. Together with the massive estimations of maintenance total costs, there is a similarly massive potential of reducing costs lying in the reduction of initial development effort.

4.7 Conclusion

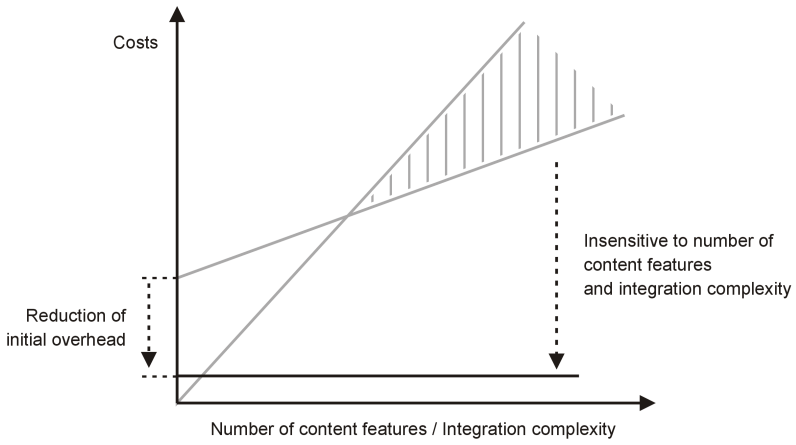

Figure 4.7: Cost requirements for implementation

This chapter introduced the different implementation models enterprise application, content application and a hybrid model implementing both application types. It also showed that the data required to be processed within the target application is mainly influencing the right model of implementation. Further it was made clear that costs in enterprise applications rise with the number of content features that have to be implemented while the costs of hybrid implementations rise proportional to the integration complexity. Additionally there are significant intial costs for setting up the hybrid environment.

The next chapter of this thesis assumes a scenario with a hybrid implementation model and a high integration complexity. Based on that a solution will be presented that applies

[13] See e.g. [BDKZ93] and [All].

the decision process and implements a hybrid model. Based on the cost behaviours presented in the previous section, the following results would be recommended as an outcome of the realization:[14]

- Lower the initial costs where possible. Some factors might not be reducable, other might be.
- Better scaling along with integration complexity. The best case would be to eliminate dependency on the integration complexity completely.
- Reduce complexity of the actual application in order to reduce maintenance costs.

[14] Figure 4.7 illustrates the requirements.

Chapter 5

Application of the Hybrid Persistence Concept and Prototypical Implementation

The previous chapter developed a decision process leading to different implementation models. The hybrid implementation was identified provide better cost behaviour on systems required to process both enterprise data and content data. Nevertheless, its initial overhead as well as the re-integration of data was identified to significantly raise the costs. This led to the requirement to implement a solution that reduces costs in hybrid implementations.

This chapter will introduce a user story to set up an example environment to serve as a demonstration ground for both the decision process and for the solution to be implemented.

5.1 User Story

The user story introduces the virtual company *Bluth Inc.*, founded by the glamourous *Bluth* family. Bluth Inc. sells different products over the web and uses a small Java-EE-based ERP[1] system for managing products and customers. As the company is having a web-based shopping platform, it wants to provide visitors with more information that just the name and the price of their products. Thus Bluth intended to maintain descriptions, technical data, the manufacturer name, a set of images and a PDF data sheet within their master data. The existing search function on the website should be modified so that search terms are also found in the new fields as well as occurences in the PDF data sheets.

As the Bluths are estimating the high value of customer retention they decided to add certain information to the master data of their customers. For example they wanted to add a free text description and a PDF profiles to be able to better keep track of customer developments.

The very same time the Bluth company decided go take the plunge and start selling products overseas. Of course this requires them to make a lot of data available in other languages. While the website itself is capable to work with multiple languages their master data is not. Hence a way should be found to make data within their ERP system internationalizable.

Gob Bluth, the CIO of the Bluth company, has mentioned that he highly appreciates efficient maintenance interfaces. Hence it should be possible to maintain all the data within the very same user interfaces. It would not be acceptable to switch between applications when maintaining master data of products and customers.

[1] Abbr.: *Enterprise Resource Planning* system.

In the near future the company also wants to join another business segment. The Bluths want to act as a content syndication provider and offer their extensive product master data to other companies. Most likely this will be realized as a web service application.

The data that are intended to be added to the application smell to a certain degree like content data. This chapter will apply the decision process to evaluate that. A hybrid implementation will be set up to add the new data. The solution developed in here will integrate both the exisiting enterprise data and the newly added content data in order to reduce costs following the requirements of Bluth Inc..

5.2 The Example Application

This section introduces the example application. Its purpose is to serve as small-scale example of a real-world application. The framework developed throughout this thesis will be integrated into the example application later in section 5.6. Although consisting of only a handful of classes it provides the architectural characterstics necessary for showing the issues of an integration.

5.2.1 Technology Prerequisites

After the basics of enterprise applications and content application were introduced in chapters 2 and 3, at this point specific products have to be chosen in order to develop and integrate concrete solutions for the given problem. Java EE 5 and Java Content Repository were chosen which the following sections are giving the rationales for.

Java EE 5 and EJB 3.0

Java EE 5 was primarily required by camunda GmbH. Nonetheless, Java EE is widely recognized as the most successful enterprise application framework of today. Java EE 5 has been introduced in section 2.5.

Java Content Repository

JCR was also required by camunda GmbH. Unlike Java EE 5, JCR didn't reach that wide level of recognition by the time of writing. Nevertheless, as it is the first and only available specification to provide a standardized API as a layer of abstraction on content repositories it was the natural choice. By using JCR, the application gets open for many products by different vendors. It also ensures longevity, a large community and fosters trust. JCR has been introduced in section 3.4.

5.2.2 Architectural Overview

The example application is built on top of Java EE 5 and EJB 3.0.[2] EJB 3 *Entities* are used to implement business entity objects. *Stateless Session Beans* are used to implement the business logic. The client application accesses the session beans over RMI to get access to entity objects. Figure 5.1 gives an architectural overview about the application.

[2]Introduced in section 2.5.

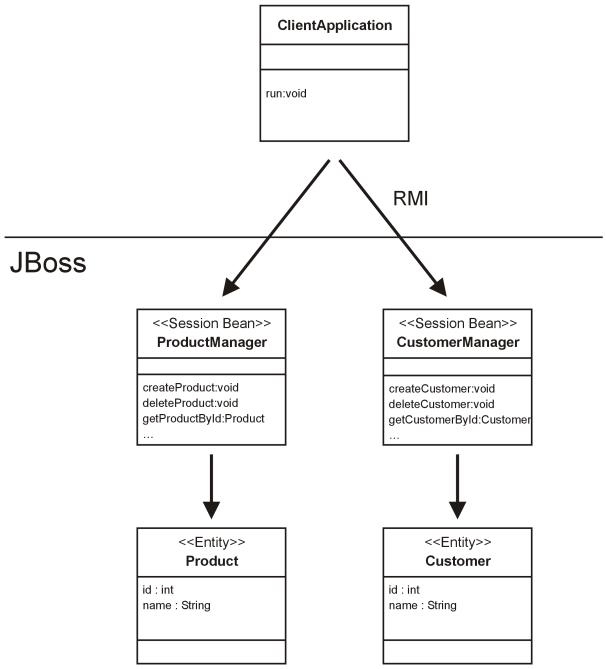

Figure 5.1: Architecture Example Application

Entities and Session Beans

The example application contains two entities, a *Customer* and a *Product*. Accordingly, two stateless session beans, *CustomerManager* and *ProductManager*, are offering methods to perform CRUD operations on the entity objects. The application is meant to offer more operations as well as rather complex business logic. For the sake of simplicity this is only indicated by empty operations *void complexBusinessOperationX()*.

Client

The client is a command-line remote application that is started from within Eclipse. It contains some use-cases to demonstrate the capabilities of the application like creating customers and products and displaying all entity objects. Later it will also serve to demonstrate the newly integrated content operations like a full-text search.

The Java source files are organized under the package *c.c.r.j.ejb3.client.**. The main method is located in *c.c.r.j.ejb.client.EjbAppClient*. The client connects to the server-side application over RMI, so RMI client libraries are necessary to be available in the runtime environment. The package included in this thesis contains a running environment including the client application and a server application on JBoss Application Server.

5.3 Decision Process Applied for the Example Application

Chapter 4 introduced different implementation models and a decision process to determine the right model of implementation depending on types of data contained in business entities of applications. This decision process is now being applied to the situation of the user story presented in section 5.1.

Survey Results - Entity 'Customer'					
	Full-text Search	Binary Search	Versioning	I18N	
id	0	0	0	0	
name	0	0	0	0	
freeTextDescription	1	0	1	1	
profile	1	1	1	0	
Total	2	1	2	1	**6**
% of all attributes	50%	25%	50%	25%	38%

Survey Results - Entity 'Product'					
	Full-text Search	Binary Search	Versioning	I18N	
id	0	0	0	0	
name	0	0	0	0	
shortDescription	1	0	1	1	
shortDescription	1	0	1	1	
technicalData	1	0	1	1	
manufacturer	1	0	0	0	
dataSheet	1	1	0	1	
images	0	0	0	0	
Total	5	1	3	4	**11**
% of all attributes	62.5%	12.5%	37.5%	50%	41%

Table 5.1: Results of the Survey Method applied for the user story situation.

The binary survey method introduced in section 4.4.1 was chosen to use for the decision process. Table 5.1 shows the results of the *survey method*[3] applied for the entities of the example application. The attributes were extracted from the user story description given earlier.

The global results can be derived from the total sums calculated during the survey analysis, which are 6 and 11. Respectively the percentages can be taken into account, which are 38% and 41% in average. In the case of this small-scale example it's even much more interesting to look at relative figures, as 6 and 11 don't sound very much. Instead of 2 entity classes with 4 and 8 attributes a real-world project might have a 100 entities with 30 attributes in average.[4] Looking at the percentages, nearly half of the attributes are content-related. This appears to be enough to decide to create a hybrid solution. Nonetheless, concrete numbers are difficult to give and small-scale examples never reflect problems very good, that tend to appear in large and complex systems. So, for this user story a hybrid implementation is chosen.

The next influencing factor, "'Implementation logic needed?"', can definitely answered with *yes*, as there is already a complex enterprise application running. In other words this means, that it is not possible anymore to choose a content-based single-system implementation, either enterprise application as a single-system or the hybrid model has to be chosen. The third group of influencing factors are "'Other factors"'. Those are ignored for now as they are hard to reproduce on a rather simple example scenario. Furthermore, they are irrelevant in this context.

Summarizing, a hybrid implementation model is being chosen for the user story situation for the following reasons:

[3] Introduced in section 4.4.1.

[4] This figure is just an example.

- Many content features required for the newly added attributes lead to a content application.
- Need for complex business logic lead to an enterprise application.

Finally the *integration complexity* of the resulting system has to be evaluated. The user story provides three factors that will increase the expected integration complexity:

- Maintenance user interfaces should be integrated. It is especially forbidden to require users to switch applications while maintaining data.
- All product master data should be published on the website.
- Content syndication system will have the need to integrate all the product data.

In conclusion a high integration complexity can be expected. On one hand this fact is legitimating to put effort into reducing the costs of integration complexity. On the other hand, the user story only fills this implementation chapter. The actual goals were given in section 4.7. According to these goals a solution will be developed to meet these goals and to represent a possible solution for the problem given in the user story.

5.4 Conceptual Overview

5.4.1 Overview

The goal is to develop a solution that allows to reduce both initial and follow-up costs when using a hybrid implementation model. This section will give a high-level overview about the approach and the different aspects. This should serve as a tool to transform the solution to other technologies.

Initial costs are the total costs that arise from the need to set up a hybrid environment. Some of these costs aren't possible to reduce, like evaluation of a content management system, licence fees, installation and administration. Others might be possible to be reduced by an appropriate integration framework, like reducing the overall complexity of the solution and reducing development costs. Thus, this frameworks aims to reduce parts of the inital costs where possible.

The second cost drivers are costs per integration complexity. In order to avoid application developers to do any further integration, an approach has been chosen that will be called *Total Integration*. This means that integration is not done on a case-per-case basis when necessary. Rather the framework will take care that in every given time, both sets of data are available, fully-integrated within the enterprise application. This approach is expected to de-couple the integration complexity and the follow-up costs during operation.

5.4.2 Conceptual Aspects

In the following, required aspects of the solution are outlined, still to some extent independent from the concrete implementation decribed later in this chapter. Figure 5.2 illustrates these aspects integrated into an enterprise application.

Content data should be generically integrated into an enterprise application. So first of all it is necessary to be able to model the content data on enterprise application level. This is not obvious as content data differs fundamentally from enterprise data. This aspect is called the **Modelling Aspect**.

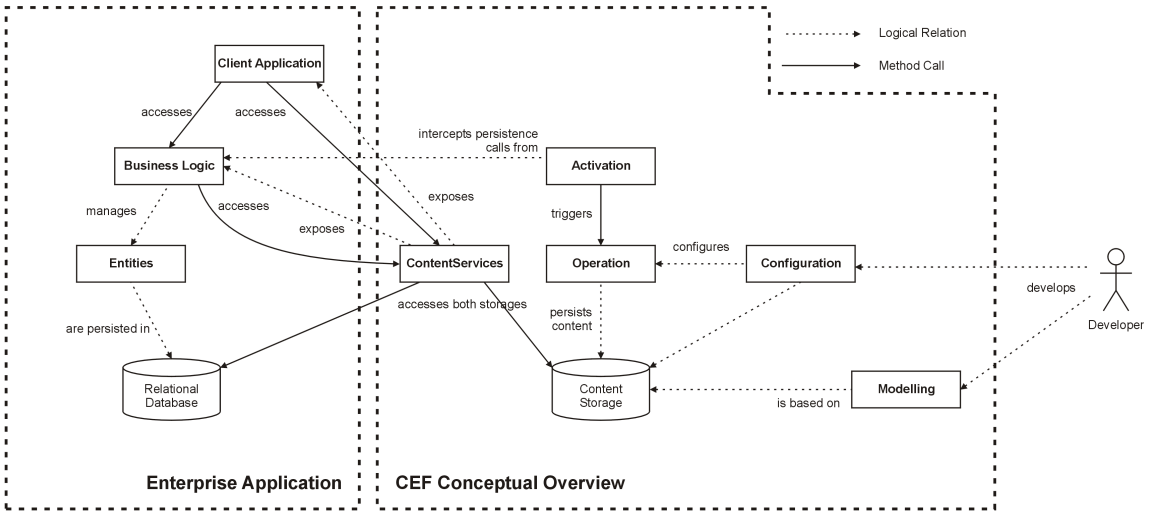

Figure 5.2: Overview Conceptual Aspects and Enterprise Application

As soon as content is modelled into the enterprise application it has to be ensured that persistence operations are always being performed on both sets of data. That way, total integration of both sets can be achieved. This is called the **Activation Aspect** as for every persistence operation that the enterprise application is performing, a second persistence operation, has to be activated.[5]

This second persistence operation for the content has to be executed on the content repository. The aspect dealing with accessing the content repository and storing, loading or deleting the content is called the **Operation Aspect**.

A configuration is necessary to achieve a well-regulated storage of the content data. This deals with structuring the content data in an appropriate way and with keeping references between enterprise data and content data in order to allow navigation in both directions between the data sets. This is called **Configuration Aspect**. It has to be noted, that configuration in this sense goes beyond usual configuration issues. It rather is a core part of the framework instead of just a helper component to configure the software.

The aspects introduced so far lead to a solution that allows to handle enterprise data and content data within the enterprise application and takes care of persisting both sets of data into two different storage systems. The last aspect is allowing to use the services of the content storage systems in order to enable the enterprise application benefit from content services delivered by the content management system. This means, that the enterprise application is for example able to perform a full-text search over a binary file. Those content-features have to be exposed to the enterprise application. This is called the **Content Services Aspect**.

During the thesis a framework called CEF, *Content Enrichment Framework*, has been developed. It implemented all these aspects according to the requirements given in the user story and to the goals proposed in the previous chapter. This framework is going to be described and explained in further detail in the next section.

[5]Actually, *Integration Aspect* would have been the name of choice, but it might have created confusion in conjunction with a lot of other occurences of the term *integration* throughout this thesis. This is why *Activation* has been chosen.

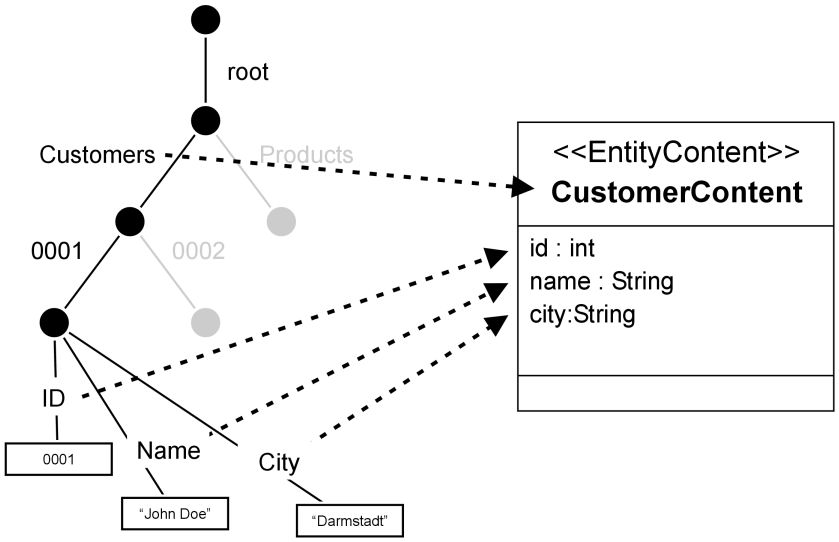

Figure 5.3: Mapping of Content Into Java Classes

5.5 Framework Implementation

The name *Content Enrichtment Framework* (CEF) has been chosen for the framework because it enriches enterprise applications with content as well as with the capability to handle content effectively. On a more technical level, it also enriches entity objects within the application with content, modelled into Java objects.

The following section is elaborating on the specific implementation. The section is organized according to the conceptual ascpect introduced earlier. Figure 5.7 illustrates the complete framework integrated into the example application.

5.5.1 Modelling

In chapters 3 and 4 the nature of content data as well as the differences to enterprise data was introduced. At first this leads to the assumption that content data can't be modelled into an enterprise application as enterprise applications are basing on fully structured data and aren't capable of efficiently handle semi-structured and unstructured data. This section will deliver a solution that is able to achieve that.

Mapping of Content Into Java

To be able to model content into an enterprise application a mapping is necessary. This mapping is done using the Apache Jackrabbit OCM component introduced in section 3.4.10. This provides a mechanism to map content stored in a JCR-compliant storage into a Java POJO model. It also provides a component to access content from within Java. Two steps are necessary:

- **POJO model:** classes represent certain units of content, e.g. *CustomerContent*. The attributes of this class are representing the values of that content, e.g. *description* or

data sheet. The data types of the attributes must be equal to the data types of the respective values of the content. Figure 5.3 gives an illustration on how content is mapped into Java classes.

- **Mapping Description:** a mapping description declares how values of the content repository should be mapped to attributes of the objects. Section B.5 shows an incomplete listing of the mapping file used in the implementation. The mapping descriptions for the CEF project can be found under *src/main/resources/cef-ocm-mapping.xml*. For content that is being directly attached to business entities the node type cef:businessEntity[6] has been defined and must be assigned within the mapping descriptions. In the example listing it can be recognized that the class-descriptor for *CustomerContent* assigns the mixin type *cef:businessEntity*.

Association between Entity and Content

After the content has been mapped into a Java object, e.g. *CustomerContent*, it has to be linked to the entity object it belongs to. This is done by creating an association between the entity object and the content object. That way, the business entity is technically implemented using two classes, the actual entity object of the enterprise application and the content object. The decision which attributes should be modelled into the content object, is directly derived from the individual results of the decision process.

Interfaces and Classes

A handful of interfaces, classes and XML descriptions, have been developed that are related to modelling and mapping content. They are described in the following sections.

Interface Content The interface *c.c.r.j.cef.impl.ocm.Content* has to be implemented in all content classes. It defines accessors for a unique identfier (String) and the path within the content repository. They are necessary to allow backwards navigation from the content repository towards the entity objects. They are also required by Jackrabbit OCM.

Class AbstractAssociatedContent The content objects do not implement the interfaces Content directly but extend *c.c.r.j.cef.impl.ocm.AbstractAssociatedContent*. This class implements the logic to hold the two values *path* and *id*.

Node Type cef:businessEntity

Node types in JCR were introduced in section 3.4.4. Node types are used to assign a certain behaviour or design to certain nodes within the content tree. In CEF, a node type is used to mark the content object as such. It is named *businessEntity*, its definition is shown in listing 5.1. It fulfills two purposes:

- It marks content within the content repository as associated to business entities. As content in JCR is organized hierarchically the content acossicated to business entity might be located in an arbitrary depth within the that content tree. And there might be content of arbitrary depth below that as well. This content *below* is illustrated in

[6] *cef* is not part of the node type name, it is used as the local namespace name. Although it should be omitted when used like it is used here but it makes it more reconizable and hence helps the reader's understanding.

figure 5.4 as *NestedContent*. The assignment of this node type to exactly that content node within the tree that is related to the business entity it makes it possible to detect that border. This is essential if content anywhere is being taken (e.g. as search result) and it has to be figured out which business entity it belongs to. Then it can be went up the content tree until the node type businessEntity is found.

- It defines the field *entityClassname*. It has to be noted that this is the class of the entity object (e.g. *Product)*, not of the entity content object (e.g. *ProductContent*). This is necessary to allow the ContentServices[7] to reversly retrieve business entities from a given content.

Listing 5.1: Node Type Definition 'cef:businessEntity'

```
<nodeType name="cef:businessEntity" isMixin="true">
  <supertypes>
    <supertype>nt:base</supertype>
  </supertypes>
  <propertyDefinition name="cef:entityClassname"
      requiredType="String"  (...) >
    <defaultValues><defaultValue /></defaultValues>
  </propertyDefinition>
</nodeType>
```

Entity-specific Content

In the example application both entities *Product* and *Customer* should be extended by data that has been identified as content. This means that Java classes for the content as well as mapping descriptions have to be created for each entity individually. For every entity that is extended with content there will be a new content class created. E.g. the class *CustomerContent* is the respective content class of the entity *Customer*.

To emphasize the independence between both classes the project stores the content classes in a different path on the file system but uses the same package structure. In production projects it is imaginable that the modelling and storage of content objects is done separately or even generated from sources fed by the content system or the like. Both entity objects and content objects do not necesarily have to be modelled at the same time and stored in the same place. Nevertheless, the content objects must be accessible by the respective entities in order to be able to create the associations between entity object and content object. This means that the content object or the respective library containing the content objects has to be available in the classpath of the entity objects.

Common Content

Besides entity-specific content there is content that might often be re-used, such as a *binary document*. It is possible and it might be recommended to re-use content objects and their mappings if possible. Those common content consist of a Java class and a XML file containing the mapping description. It is imaginable that these bundles of common content are managed and shared centrally within an organization. In this thesis that is outlined by two common sets of content, *ImageSet* and *BinaryDocument*. They can be found under the package structure *c.c.r.j.cef.impl.ocm.common*.

[7] See section 5.5.5.

ImageSet *ImageSet* contains images of the different sizes small, medium, large and press. They might be used for products, where images for different places of usage are necessary, e.g. a small picture for product lists, a large picture for a detailed view and a very large press picture for publishing purposes. In the example application this common content is used for storing pictures of the products.

BinaryDocument The *BinaryDocument* can hold documents of any binary format like PDF, Excel or JPEG. Additionally this content defintion defines meta-information like the mime type or the date of the last modification. In the example application this common content is used for storing any binary data (except for product images) like the data sheet of *Products* or the profile of *Customers*. In Java then, common content can simply be used as shown in listing 5.2.

Listing 5.2: Usage Example Common Content

```
1 public class CustomerContent {
2
3   private BinaryDocument profile;
4
5   ...
6 }
```

Problems and Pitfalls

The advantage of this approach is that it makes data available in Java objects, that is actually stored in a content management system. That way most of the issues of the integration of enterprise data and content data can technically be solved in Java.

Nonetheless there is one major downside of this approach. The semi-structuredness and unstructuredness of the content gets lost to some degree in the moment Java classes and mapping descriptors have to be created. This also steals much of the flexibility that is typical for content and content management systems. On the other hand it is often advisable to have fully-structured and pre-defined data at hand in many cases. If for example a picture should be displayed on the web then it might be helpful if the web application can rely on the existance of a certain attribute denoting the actual image.

Depending on the case, it might be helpful to have a bit more unstructuredness at hand. Possible ways to achieve that are:

- Load the content properties into dynamic maps as key-value pairs instead of static attributes.
- Use a mechanism that allows to dynamically generate or change classes. AspectJ for example provides this feature called *Introduction*.[8] A different approach, amongst others, would be to use a mechanism called *DynaBean* from the Apache Commons BeanUtils project.[9]
- JCR itself provides a sohphisticated API to access unstructured data in a uniform way. While there are some technical issues prohibiting to easily expose that API to remote clients, local clients like web applications could use this API under certain conditions.

[8] See [Lad03], page 35.

[9] More details can be found on the Apache Commons BeanUtils homepage under *http://commons.apache.org/beanutils*.

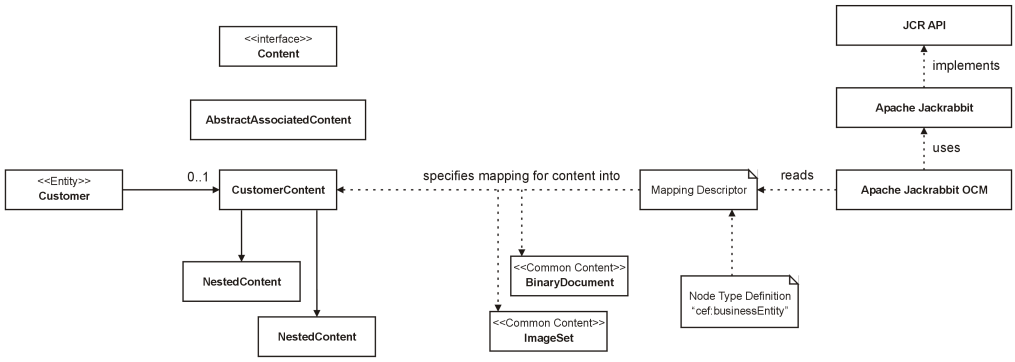

Figure 5.4: Implementation Overview of the Modelling Aspect

In the context of this thesis Jackrabbit OCM was chosen as it was publicly available in a stable release by the time of writing. Due to the modularization into different aspects, this component could be exchanged using another technique of modelling and mapping.

5.5.2 Activation

Overview

In CEF *Activation* is being implemented using EJB 3.0 interceptors[10]. Section 2.3.3 introduced the four basic persistence operations create, read, update and delete. Accordingly, different types of interceptors have been implemented to realize activation for those operations:

- PersistInterceptor
- PersistCollectionInterceptor
- LoadInterceptor
- LoadCollectionInterceptor
- DeleteInterceptor

Persistence operations for content should be performed synchronously to persistence operation on the entity object. This is an essential part of meeting the requirement of a total integration. As those persistence operations on the entities take place in the session beans, the interceptors have to be registered on the session beans, too. For every method a persistence operation takes place (e.g. *createCustomer()*) one appropriate interceptor has to be registered.

First, the kind of persistence operation according to the CRUD model has to be determined. The method *createCustomer()* for example performs a *Create* operation. The right interceptor has to be chosen according to the mapping given in table 5.2.

The declaration "'Single Value"' or "'Collection"' and hence also the type of interceptor is determined by the entity being persisted. If a single value entity object is given as parameter then this is referred to as a "'Single Value"'. If a collection of objects is given as parameter[11] it is referred to as "'Collections"'.

[10] Activation was introduced in section 5.5.2, interceptors in section 2.5.5.

[11] This is also known as *bulk* operations.

CRUD Operations - Interceptor Types								
	Single Values				Collections			
	C	R	U	D	C	R	U	D
PersistInterceptor	X		X					
PersistCollectionInterceptor					X		X	
LoadInterceptor		X						
LoadCollectionInterceptor						X		
DeleteInterceptor				X				N/A[12]

Table 5.2: Mapping between CRUD types and interceptor types

All interceptor classes belong to the package *c.c.r.j.cef.impl.ocm.ejb3interceptor*. The integration of interceptors into the example application is described later in section 5.6.3.

Implementation

Implementation of all interceptors is following the same operational sequence. Calls from the client on the session beans are being intercepted by the respective interceptor. Then the actual call to the session bean method is done. Afterwards the call to the content manager, introduced later in section 5.5.4, is done. Within the scope of the interceptor both the parameters passed to the session bean operation as well as the return value from the session bean operation are available. These values are then passed to the content manager which then performs the actual persistence of the content associated with the entity object. This call varies depending on the interceptor type as follows:

- **Persist*Interceptor:** It is expected that the entity / entities to persist are passed as parameter of the session bean method. The interceptor is reading this parameter, extracting it and passing it to the content manager. In case of the PersistCollectionInterceptor, a collection of entities is expected as parameter.
- **Load*Interceptor:** It is expected that the entity / entities to load are returned by the session bean method. The interceptor is reading the returned object and passing it to the content manager. The content manager then loads the respective content, attaches it to the given entity and returns it back to the interceptor. The interceptor is then returning that object.
- **DeleteInterceptor:** It is expected that the entity class as well as the identifier are passed as parameter to the session bean method. The interceptor is first calling the session bean method in order to delete the entity object. Then, class and identifier are passed as parameters to the content manager which then deletes the content. It has to be noted that due to technical problems no delete interceptor has been implemented that accepts collections. It has been omitted also due to the fact, that in practice it is not a common use-case to remove collections of entities. If necessary, this feature would have to be realized outside this thesis.

5.5.3 Configuration

The *configuration aspect* is responsible for providing a configuration mechanism. In CEF this is done by the interface *com.camunda.research.jcr.cef.impl.ocm.ContentConfiguration* and its implementation *com.camunda.research.jcr.cef.impl.ocm.AnnotationContentConfiguration*.

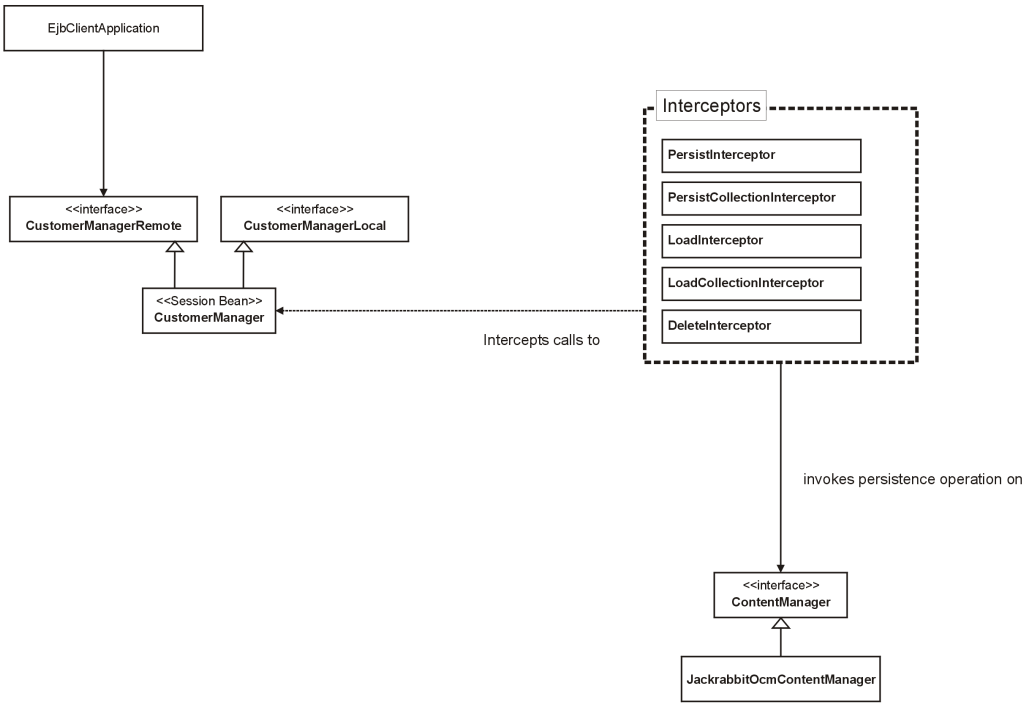

Figure 5.5: Implementation Overview of the Activation Aspect

Interface ContentConfiguration

The interface ContentConfiguration provides methods that return configuration data. These data are mostly used by the content manager. Some of these data are global settings, e.g. the root directory of all CEF-related content data within the content repository. Others are related to entities and expect the entity object as parameter. This interface can be found in listing B.7.

Class AnnotationContentConfiguation

The implementation of ContentConfiguration interface uses Java annotations[13]. Except for some settings, that are static values within the class definition, all settings are done by a set of custom annotations. The *AnnotationContentConfiguration* class is reading those annotations and returning the settings according to annotations set within the classes. The usage of these annotation is explained in further detail in section 5.6 about integrating CEF into the example application.

Content Path Configurations

A major responsibility of the configuration aspect is to configure how the content is being structured within the content management system. As content stored by JCR-compliant repositories is organized hierarchically[14], a hierarchy structure has to be assured that is both consistent for each content as also intuitive and human-readable[15]. Content structures are

[13] Introduced in section 2.5.6.
[14] See section 3.4.3.
[15] See section 3.2.2 for the meaning of human-readable content structures.

built of three different parts:

1. **Root Path** This is the absolute root path under which all the content persisted by CEF is collected. This is denote by the constant *AnnotationContentConfiguration.REPO_ROOT*. The root path is only used for building other parts of the path.

2. **Collection Path** This path is right under the root path and is unique for every entity. It can be manipulated by the annotation *@ContentPathPattern*, explained in section 5.5.3. The default value for this is the full-qualified class name of the entity class, the content is associated with.

3. **Entity Path** This path is right under the collection path and is unique for every instance of a entity. The business identifier, determined by the annotation *@ContentId* which is explained in section 5.5.3, is used to build this path. Under the entity path, all the content related to the respective entity is stored.

Listing 5.3 is showing an example path configuration. */root* denotes the root path of CEF. */root/Customer* and */root/Product* are the collection paths of the entites Customer and Product. */root/Customer/001* is an example entity path of the business entity Customer. The last element of this entity path is build from the identifier of the entity object. In the example project this identifier *001* would be created by the JPA entity manager. As explained in section 5.5.2, the relational persistence operation of the entities are taking place before the content persistence operation. In case a new entity object is created, first of all the entity manager is creating this identifier. It hence takes care of unique identifiers for the entities. The content persistence is then taking this identifier and using it for persistence of the content. Thus, an assocation between both, the enterprise data and the content data, is established.

Listing 5.3: Example Content Structure Resulting from Content Configuration

```
/root
  /Customer
    /001
      + freeTextDescription
      +...
    /002
  /Product
    ...
```

Annotation @ContentEnriched

This annotation is used to mark entity objects that are enriched by content. This means that those entities will be processed by the content manager component. This annotation doesn't have any further parameters or attributes.

Annotation @ContentPathPattern

This annotation is used to configure the collection path. This can either be done by free text denoting the actual names of the desired path, e.g. *People/Customers*. Additionally, two variables can be used here:

- **%FCQNAME%:** Denotes the full qualified class name of the associated business entity. This can be used for creating flat content hierarchies by having unique identifiers for each entity object under the same node.

Listing 5.4: Flat Content Structure

```
Created by content path pattern: \%FQCNAME\% (default pattern)

/ExampleApplicationRoot
  /com.company.Customer
    /001
    /002
    /003
    /...
  /com.company.Supplier
    /...
  /com.company.Product
    /001
    /002
    /...
```

Listing 5.5: Semantically Grouped Content Structure

```
Created by content path pattern: People/\%SNAME\% or e.g. People/Customer, People/Supplier

/ExampleApplicationRoot
  /People
    /Customer
      /001
      /002
      /003
      /...
    /Supplier
      /...
  /Things
    /Product
      /001
      /002
      /...
```

- **%SNAME%:** Denotes the simple class name of the associated business entity, i.e. only the actual class name without the package names.

This annotation is optional. If no path pattern is given, the default value will be the same as the variable *%FCQNAME%* resulting in a flat hierarchie under the root node. This is done because it ensures unique paths for each entity. As it can easily lead to unclear structures (e.g. if there are many entities), it is recommended if either paths should not necessarily be human-readable of if the number of entities is relatively low. An example resulting content structure is shown in listing 5.4. By giving path patterns that are doing a grouping the resulting paths get easily human-readable and can be maintained clearer even with a large number of entities. An example for a grouped content structure is given in listing 5.5.

Annotation @ContentId

This annotation is used to mark which field of the entity should be used for uniquely identifying the content. This id is primarily used for two purposes:

- Creating the last piece of the content path, the entity path. In listing 5.4 it can be recoginized as *001*, *002* and so on.
- Creating references between the entity and the associated content in order to be able to navigate in both directions between entity data and content data.

Listing 5.6: Versioning Implementation in JackrabbitContentManager

```
boolean checkedOut = false;
if (objectContentManager.objectExists(content.getPath())) {
  try {
    objectContentManager.checkout(content.getPath());
    checkedOut = true;
  } catch (VersionException e) {
    logger.debug(content.getPath() + " is not versionable apparantly.");
  }

  objectContentManager.update(content);
} else {
  objectContentManager.insert(content);
  RepositoryHelper.setProperty(session, path, "cef:entityClassname", entityClassname);
}
objectContentManager.save();
if (checkedOut) {
  objectContentManager.checkin(content.getPath());
}
```

Annotation @ContentAssocationGetter

The content object is attached to the entity object by association. The content manager needs to extract content object from the entity object. This annotation marks the method that allows to retrieve the content object, e.g. *public CustomerContent getContent()*.

Annotation @ContentAssocationSetter

Similar to the previous annotation, it marks the method that allows to set the content object, e.g. *public void setContent(CustomerContent cc)*.

5.5.4 Operation

The *operation aspect* is responsible for actually performing the persistence operations of the content attached to entity objects. The core of this aspect is the definition of a *ContentManager* interface providing basic persistence operations, similar to JPA's entity manager[16], and an implementation of this interface which is based on JCR and Apache Jackrabbit[17]. The execution of these operations on the content manager is triggered by components implementing the *activation aspect*, described in the previous section.

Interface ContentManager

ContentManager is the sole interface between the target application and CEF. It is defined as part of the project CEF as *c.c.r.j.cef.impl.ocm.ContentManager*. It defines the following methods:

- persist()
- persistCollection()
- load()
- loadCollection()

[16]Introduced in section 2.5.3
[17]Introduced in section 3.4

- remove()

Obviously there are similarities between the interceptor types and the persistence operations offered by the content manager. Unlike the JPA entity manager, the content manager doesn't distinguish *create* and *update* methods. This isn't necessary as no attention has to be paid to the existance of content within the content repository. Roughly speaking, content just gets persisted, no matter if it had been existed beforehands or not. Existing content is being overwritten. This behaviour will be changed if content is versionable. In that case, content will not be overwritten but new versions are created.[18] This behaviour could also be changed if necessary in certain contexts.

Furthermore the interface doesn't define any query methods, also opposite to the JPA entity manager. Querying operations are fully implemented in an external service introduced later in section 5.5.5 as part of the *content services aspect*.

Class JackrabbitOcmContentManager

The concrete implementation of the ContentManager interface is *c.c.r.j.cef.impl.ocm. JackrabbitOcmContentManager*. In order to be able to make use of some convenient features of EJB 3 such as dependency injection, it is declared as a stateless session bean. The ContentManager interface acts as the local interface. A remote interface isn't offered as the content manager is not intended to be used directly by clients.

The content manager is using Jackrabbit OCM[19] for persisting the content objects. In the constructor of the content manager, Jackrabbit OCM is being loaded. At this point it loads the current mapping descriptors of entity content objects as well as common content objects. This is illustrated in listing 5.7.

Listing 5.7: Jackrabbit OCM loads mapping descriptors

```
1 InputStream mapping = this.getClass().getResourceAsStream("/cef-ocm-mapping.xml");
2
3 this.objectContentManager = new ObjectContentManagerImpl(session,
4                                                  new InputStream[] { mapping });
```

The important information about content to store, load and remove Jackrabbit OCM needs to know is the path and the identifier. The identifier doesn't necessarily has to be the same as the identifier of the entities but in CEF the same identifiers are used. This means, that the same identifiers are used for enterprise data in Java EE as well as the associated content in JCR. This allows navigation from the content towards the entities. This features is used for the content services later in section 5.5.5. The necessesity that information about the path and the identifier have to be stored is also the reason why it has been defined for the interface *Content*, introduced in section 5.5.1. This way, every content carries attributes for the path and the identifier.

The client doesn't need to know about paths and identifiers. Especially it doesn't have to set these values, as the content manager is taking care of this. It uses the content configuration object to determine path and identifier for every given entity. It has to be kept in mind, that the parameter passed to the content manager is only of the type *Object* and represents an entity object. Attached to that entity might be a content object. And this content object might have further nested content, e.g. associated with it. The basic workflow for every persistence operation is the following:

[18] See the following section for the implementation of versioning.
[19] Introduced in section 3.4.10.

1. Check whether the given entity is marked with *@ContentEnriched*. This determines whether it is possible at all that this entity might carry content.
2. Check whether a content object is attached. It might be possible that the entity could carry content, but in fact it does not. This is loosely related to a characteristic of semi-structured and unstructured content given in section 3.1. But it's also reasonable programming practice to perform this check.
3. Determine the path of the content associated with the entity and inject it into the content object. On persist operations, the path is needed to store the content. On retrieve operations the path is needed to load the content from. This is done using the content configuration.
4. Determine the identifier of the given entity and attach it to the content object via injection.
5. Perfom the actual persistence operation. As the two necessary pieces of information, path and identifier, have been injected into the content object, it is now possible to persist the content object independently from the entity.
6. On *retrieve* operations, the retrieved content object is injected into the given entity. Afterwards the entity object is returned.

Versioning is one of the content features listed in section 3.2. Versioning is the only one of the given content features that is directly implemented into the content manager. To enable versioning for a given content, the proper mixin node type has to be set in the mapping description. If the content manager realizes that a given content is versionable, it performs the appropriate versioning commands automatically. This is illustrated in listing 5.6. It has to be noted that in the implementation of this thesis, versioning is only applicable for a whole content object. The JCR API offers versioning on individual properties. This is also the way versioning is described earlier, e.g. in the description of the survey method in section 4.4.1.

A full, but cleaned-up, listing of the JackrabbitOcmContentManager class can be found in listing B.8.

5.5.5 Content Services

The previous section described the implementation of the aspects that allow a total integration of content into an enterprise application. This section describes the implementation of the content services aspect, which exposes content features like full-text search[20] to the enterprise application. This last step is necessary so that the enterprise application can benefit from features that the content application delivers out-of-the-box. It has to be noted, that some of the content use-cases are weaved into the operation aspect, such as versioning, or have to be implemented using the modelling aspect, such as internationalization[21]. The content services are generally responsible for querying operations on the content.

Conceptual Overview

The basic is that content use-cases are performed by the content application and the content services, as a part of CEF, are exposing these features, or interfaces to perform these features respectively, to the enterprise application.

[20] The content use-cases are introduced in section 3.2.
[21] Internationalization isn't shown in this thesis.

Therefore the content services are delegating content-related requests to the content application. They are able to access the content repository as well as the entity manager of the enterprise application. Hence they are able to e.g. search for content and then additionally load the respective entity objects in order to load the entity object and attach the content. This allows a reverse access to the entity objects. This mechanism realized the total integration for the content services.

Session Bean 'ContentServices'

The actual service that is offerering those content services is implemented as a stateless session bean in *c.c.r.c.cef.reverse.ContentServices*.

Dependencies ContentServices has to have access to the following components:

- **JPA EntityManager** It has to have access to the same entity manager that is reponsible for persisting the entities of the application. This is important for the service to be able to load existing entities. In the prototype this is done by just putting the application as well as the CEF framework into the same .jar file. This implicitly forces EJB to inject the same entity manager when using the @PersistenceContext annotation. Please note, that in a real project it would be impracticable to put both the application and the framework into the same jar. Here it has been done for the sake of simplicity.
- **Content Repository** It also has to have access to the content repository where the content belonging to the application is stored. In the prototype this is done by using the same lookup code or the same JNDI name respectively for the JCR session object.

Implementation The session bean contains helper methods as well as the actual services. Most notably among these helper methods are:

- **Node getEntityRootNode(Item item)** This method recursively climbs up the content tree starting at the given JCR item until the marker node type *cef:businessEntity* found. It then returns this content object.
- **Node getAssociatedContentForEntity(String path)** This methods returns the same value as the previous operation but for a given path instead of an JCR item.
- **String getEntityIdForItem(String path)** This returns the entity ID of the content stored under the given path. It loads the content under that path, reads the property, that carries the identifier, and returns its value. It is used for determining which entity instance has to be loaded from JPA for a given piece of content.
- **Object getEntityContentBundle(String path)** This loads the content at the given path, extracts the class of the entity it is associated with and extracts the identifier of the entity instance. With this data it is able to load the entity from the JPA entity manager. Finally it injects the content object into the entity and returns the entity. This methods is fundamental for other content service operations.

During the thesis only one content operation has been implemented. Listing 5.8 shows the source code of this operation. In line 6 an XPATH query is created containing the query string. This query is delegated to JCR in line 9. This part is essential, as it is visible that the whole complexity of a full-text search is available by just using a handful of lines of code. After having put a lot of initial effort into the hybrid system, this is the moment where it

Listing 5.8: Implementation of ContentServices.fullTextQuery(String queryString)

```
public Collection<Object> fullTextQuery(String queryString)
      throws ItemEntityAssocationException {

  Collection<Object> result = new ArrayList<Object>();

  String xpathQuery = "//*[jcr:contains(., '" + queryString + "')] ";

  try {
    Query query = session.getWorkspace().getQueryManager().createQuery(xpathQuery,
                                                                       Query.XPATH);
    QueryResult queryResult = query.execute();
    NodeIterator it = queryResult.getNodes();
    Node node;
    while (it.hasNext()) {
      node = it.nextNode();
      result.add(getEntityContentBundle(node.getPath()));
    }
      return result;
    } catch (Exception e) {
      throw new ItemEntityAssocationException("Error on full-text search: '" +
                                                          queryString + "'", e);
  }
}
```

pays back. Every single feature that the content repository, or JCR respectively, is able to perform can be exposed to CEF, and hence to the target application, by such an operation!

After JCR returns matches for the XPATH query, the session bean uses the helper methods mentioned above to retrieve the associated entities and to return them as a bundle.

Usage Example If using this service from a client the following code makes clear that it's possible within one method call to perform a content operation and get the entity-content bundle back. This is a major saving in terms of lines of code and network calls compared to a manula integration where typically two network calls have to be done. Listing 5.9 shows how to use the full-text query operation in a client application:

Listing 5.9: Usage example of ContentServices.fullTextQuery(String queryString)

```
ContentServicesRemote contentServices;
// Lookup code here or using dependency injection
Collection<Object> result = contentServices.performFullTextQuery("Darmstadt");
for (Object o:result) {
  System.out.println(o.getClass());
  // Do something else
}
```

5.6 Integration in the Example Application

This section describes the integration of the framework developed during this work into the example application that was introduced at the beginning of this chapter. First the prerequisites on the environment is shown. The example application itself requires a running application server environment according to the requirements of Java EE 5 and EJB 3 but the extensions that are being integrates do have some further requirements. Then modifications on entities as well as on session beans are explained. Finally transactions are taken into account. Figure 5.6 is showing the example application after content has been integrated.

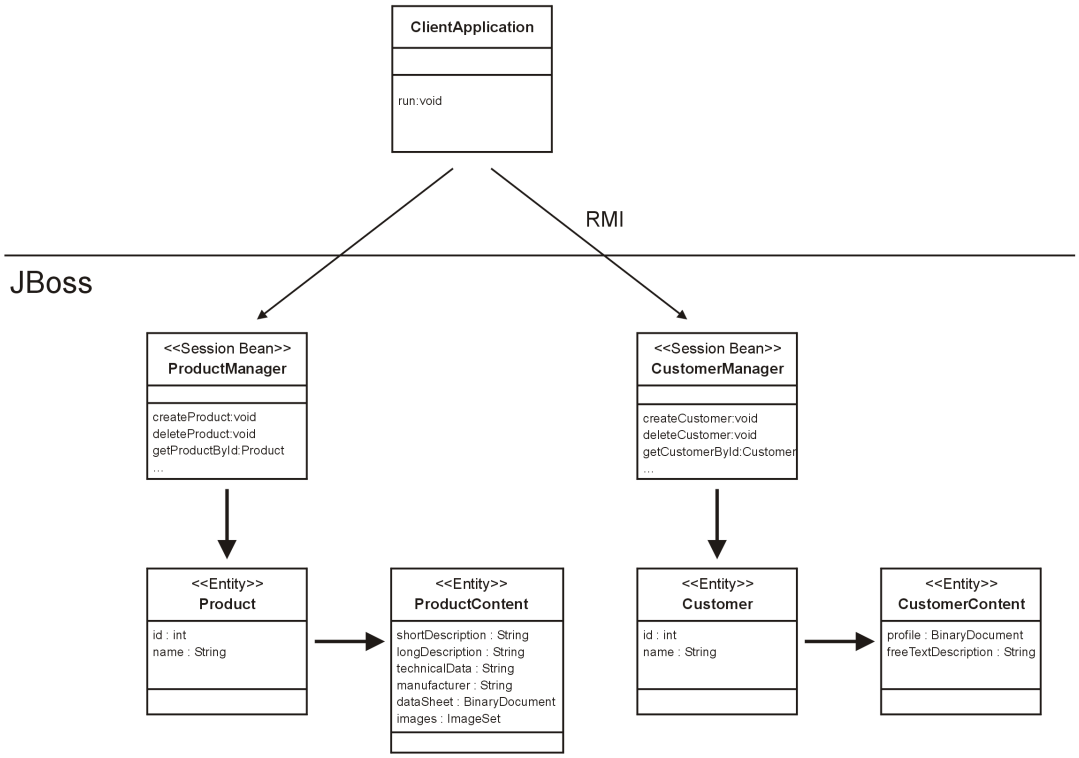

Figure 5.6: Architecture Example Application with Content

5.6.1 Environment Prerequisites

The most important part of the environment to set up is a JCR-compliant content repository. In this thesis Apache Jackrabbit[22] is being used. Hence a running Jackrabbit instance has to be set up and registered in JNDI.[23].

5.6.2 Modifications on Entities

In CEF the content objects are attached to the actual entities. The necessary modifications are described in the following. An example entity object can be found in section B.2.

1. **Mark Entities as ContentEnriched** Entities that are carrying content first have to be marked by the annotation *ContentEnriched*[24]. This allows CEF to efficiently recognize whether an entity carries content. As it is only a marker annotation[25] no more parameters have to be set.

2. **Meta Information For Content Structuring** CEF has to be provided with meta information about how to structure the content. As explained in section 5.5.3, there are default values but the structure can also be customized. This is done by using the class-level annotation *@ContentPathPattern("People/Customers")*, as explained in section 5.5.3. The given path, here "'People/Customers"' as an example, denotes the root path of content belonging to that entity.

[22]Introduced in section 3.4.9.
[23]Instructions can be found at *http://jackrabbit.apache.org*.
[24]See section 5.5.3.
[25]This is not an official term but it is inspired by the phrase *marker interface*.

3. **Denote getter and setter for content object** As explained in section 5.5.3 and 5.5.3 the getter method for the content object has to be marked by *@GetAssociatedContent*, the setter method has to be marked by *@SetAssociatedContent*.

4. **Mark attached content object as transient** In order to avoid that the content object is being persisted, also these accossor methods have to be marked as transient by *@javax.jpa.Transient*[26].

5.6.3 Modifications on SessionBeans

CEF as it is implemented requires that persistence services are exposed by session beans.[27] These are the points in the source code where persistence operations for the actual entities are initiated. Here, also the persistence operation of the content must be triggered. Thus some changes to the session beans must be done which are explained in the following.

CEF includes a set of interceptors that allows simple integration of the activation aspect. According to section 5.5.2 the right interceptor has to be chosen for each session bean operation where persistence operations are performed. The actual registration of interceptors has been described in section 2.5.5. Listing 5.10 shows an example:

Listing 5.10: Example Interceptor Registration

```
1  @Interceptors(PersistInterceptor.class)
2  public void createProduct(Product p) {
3    entityManager.persist(p);
4  }
5
6  @Interceptors(LoadCollectionInterceptor.class)
7  public Collection<Product> getAllProducts() {
8    Query query = entityManager.createQuery("select p from Product p");
9    Collection list = query.getResultList();
10   return list;
11 }
```

5.6.4 Transactions

As introduced in section 2.3.5 persistence operations like the one here must be executed transactionally. As there are two distinct persistence operations for the same entity it is important to ensure that both operations succeed or none. Therefore a transaction is required that spans both operations. Thus, the two following things have to be done:

1. Make sure a superior transaction is spanning both operations.

2. Make sure any failure occuring in one of the two operations will roll back the transaction properly.

Spanning exception over both persistence operations

Spanning transactions must be initialized before calling the two operations. As the session beans of the application are the components calling the persistence operations, it must be

[26] It is clear that this annotation is provided by JPA as well as used by JPA. It is not part of the CEF framework.

[27] This must not be the case nor is it meant as a general design advice. See section 6.4 for further information about applicabilty for other platforms.

ensured within here. Eventually a transaction might be existing even before the session bean operation is called. In this case, the surrounding transaction should be used. In EJB 3.0 the most convenient way to archive this is by giving transaction responisiblity to the application server by setting the annotation

@javax.ejb.TransactionManangement(TransactionManagementType.CONTAINER)

Furthore EJB 3.0 provides flexibility on what to do with surrounding transcations and when to start new transactions. This can be done by using the *javax.ejb.TransactionAttribute* annotation. In this thesis the proper transaction attribute had been chosen to be

@javax.ejb.TransactionAttribute(TransactionAttributeType.REQUIRED)

Rollbacks on Exceptions

In case of any exception occuring within each of the persistence operations the spanning transaction must be rolled back. For the JPA operation no special treatment is necessary, as transaction handling is integrated properly into EJB. But still, transaction handling for the the content persistence operation has to be ensured. The right handling would be ensure that the spanning transaction is being rolled back on any error occuring within the content manager. Thus, the content manager is throwing either one of two different exceptions created for this purpose:

- *c.c.r.j.cef.impl.ocm.ContentPersistenceOperationFailedException*
- *c.c.r.j.cef.impl.ocm.ContentStorageException*

Both exceptions are derived from RuntimeException. As stated earlier in section 2.3.5, EJB is specified to roll back transactions automatically, if RuntimeExceptions are being thrown.

Additionally, these two exceoptions have been annotated with *javax.ejb. ApplicationException(rollback=true)*, which also makes sure that this particular exception rolls back transactions.[28]

5.7 Future Work

This implementation of CEF is functional and gives a clear demonstration the total integration principle. Yet its development was heavily constrained by time having the need to fit into this thesis. In the following, some thoughts about the future work of CEF are presented. It has to be noted, that this refers only to the framework CEF. A general section about future work can be found in the end of this thesis in section 7.1.

- CEF performs the total integration on every persistence operation without any exceptions. In practice this would mean, that often large binary objects would be loaded that don't necessarily have to be loaded. A solution would be to implement a partial loading or lazy loading mechanism on content.

[28] As the exception is derived from RuntimeException, this would not have been necessary anymore. But it also doesn't have any negative effects.

- The modelling aspect only allows to have content attached to an entity. This implies that no content object could be created stand-alone. This would also imply that entities are always implemented as EJB entities without any regard to the individual results of the decision process.
- Implement other methods of modelling and mapping, as already proposed in section 5.5.1.
- Instead of the *AnnotationContentConfiguration*, a configuration based on XML would avoid the neccesity to alter the source code if changes to CEF configurations would have to be made.
- Although there are no major performance issues expected that are due to the source code of CEF, extensive performance test under various conditions would be recommended.
- JPA is able to persist graphs of entities. CEF expects the entity given as parameter at the session bean operation and doesn't recognize content attached to nested objects.
- CEF as well as the example application is being deployed as part of the same enterprise archive. A single deployable package of the CEF code should be created that can better be shared and deployed.

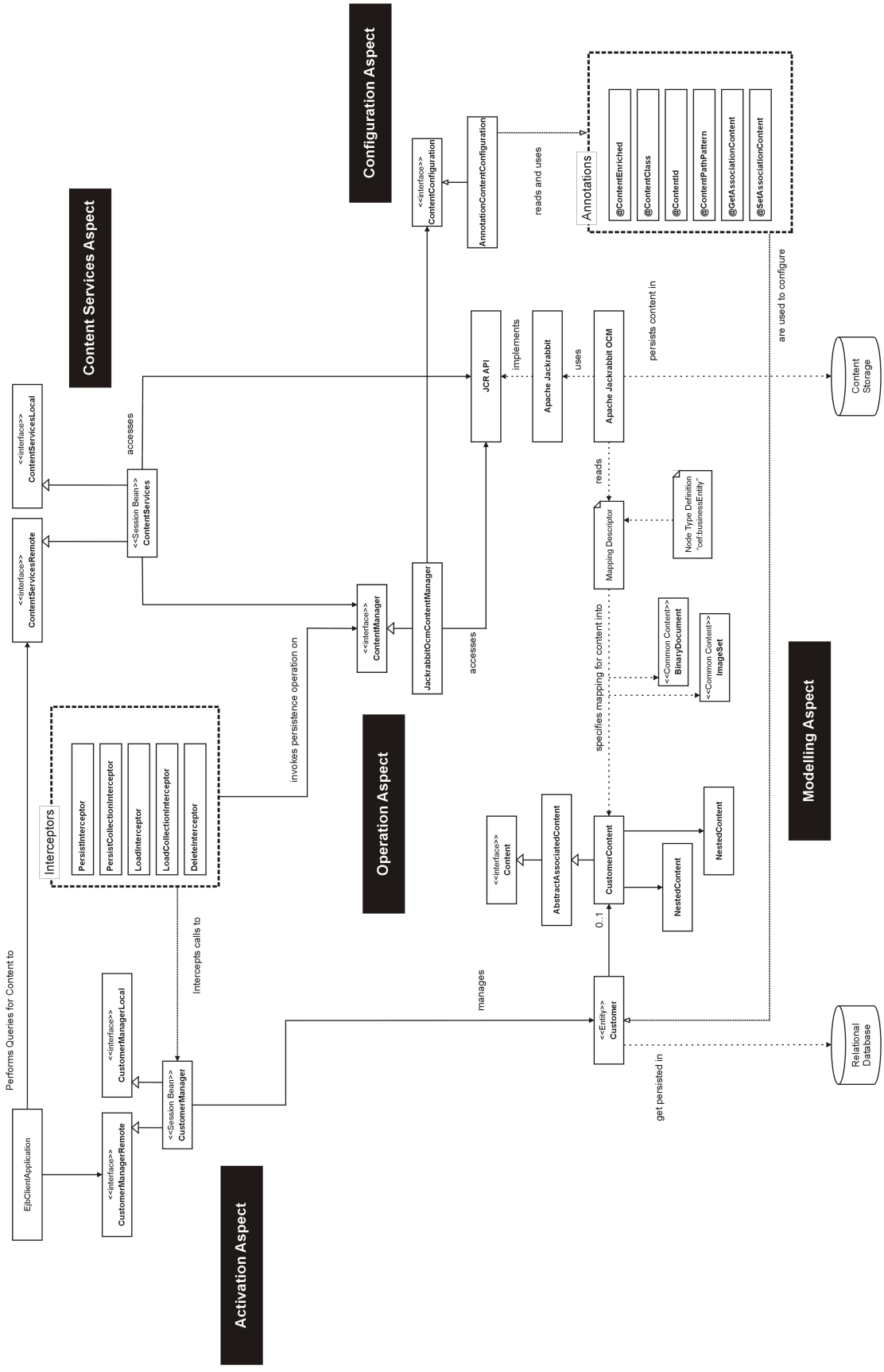

Figure 5.7: Overview Diagram of all CEF Components Integrated into the Example Application

Chapter 6

Evaluation

The following chapter evaluates the value of this thesis. To achieve this, different aspects are taken into account:

- First, to verify that scope and the overall goals of the thesis, introduced in section 1.2 have been met. They required to develop a decision process leading to concrete implementation models and to investige on cost behaviours on different models. Furthermore they required to implement a concrete solution reducing costs in hybrid implementation models. These goals have been refined in section 4.7.
- Furthermore, the applicability of the thesis for other technologies and platforms should be taken into account as a wider range of applicability implies a greater overall value of the results of this thesis.

The following sections will take a look at the different aspects in order to show, that the goals of this thesis have been reached. Therefore, the decision process, the implementation and the applicability will be investigated.

6.1 The Decision Process

Chapter 4 introduced the decision process. Delivering global results and individual results, this process supports to determine the appropriate model of implementation and also supports in separating the data in case of hybrid implementation. Section 5.3 also applied the decision process on the example application. Future work on this process will be presented later in section 7.1. During the elaboration on the decision process throughout chapter 4 all factors influencing this process are introduced. Also, all different implementation models were explained which are the outcomes of this process.

6.2 Fields of Suitability

The decision process presented in chapter 4 leads to two different architectural choices: a single-system implementation model or a hybrid system implementation model. Furthermore it coined the term *integration complexity* in order to explain the behavior of follow-up costs of a hybrid system. Chapter 5 then provided an implementation of a framework that provided a *total integration* on enterprise application level for hybrid implementations. This total integration is reducing follow-up costs. The savings get the more significantly the higher the integration complexity is. The diagram in figure 6.1 illustrates this. The left half of the

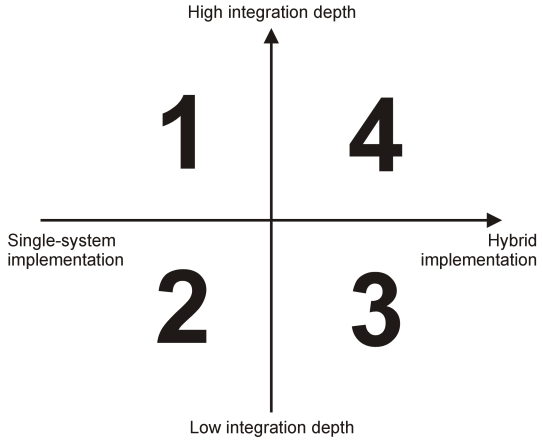

Figure 6.1: Fields of suitability for CEF

matrix is representing a single-system implementation, thus the integration complexity does not apply. The right side represents a hybrid system. Quadrant 3 shows hybrid systems with a low integration complexity. Here, an integration on a case-per-case basis might be suitable. In quadrant 4, hybrid system with high integration complexities, the follow-up costs are rapidly increasing. These are the cases where it is suitable and recommended to use CEF to achieve cost reductions.

6.3 Cost Savings

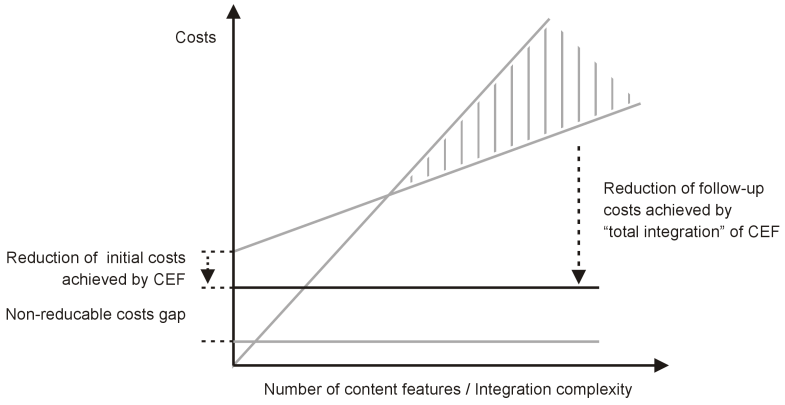

Figure 6.2: Cost savings for hybrid implementations achieved by CEF

Figure 4.6 in section 4.6.3 illustrated the cost behaviour of hybrid implementation models with manual integration. This combination doesn't scale well with high integration complexities. Figure 4.7 in section 4.7 illustrated the desired cost savings: the reduction of initial costs as well as the reduction of costs associated with high integration complexities. Figure 6.2 illustrates the cost savings achieved by CEF. The following sections elaborate on the two different kinds of costs and the savings that could be reached.

6.3.1 Follow-up Costs

The approach of a *total integration* as explained in section 5.4 makes further manual integration tasks unnecessary and obsolete. This de-couples follow-up costs and integration complexity as shown in figure 4.6.

6.3.2 Initial costs

The following list according to section 4.6.4 gives details about costs saving on individual initial cost factors.

- **Content Application Environment** This factors can not be reduced by CEF.
- **Developer Know-How** These initial costs can be reduced by CEF. CEF is hiding a large part of the integration complexity. It also allows to model content using an annotated POJO model. This only required little new knowledge about issues of an integration.
- **Time-to-market** As CEF is hiding much of the integration issues and simplifying the complexity the time necessary to build hybrid implementation has been reduced significantly. This also includes prototypes that are necessary in early project phases.

6.3.3 Maintenance Costs

CEF is designed to be universally integrateable into enterprise applications compliant to Java EE 5. This implies that it increases the complexity of the actual enterprise applicaiton only to a minimal degree. The largest part of the complexity to handle the hybrid persistence is implemented outside the application. According to 4.6.5, this reduces the maintenance costs in comparison with a manual integration within the sources of the enterprise application. Although concrete numbers are difficult to estimate in general, facing the big amount of costs necessary for maintenance, this is a big improvement over manual integration.

6.4 Applicability for other platforms

The prototypical implementation of the hybrid persistence concept has been tailored to a Java EE 5 environment using JCR for accessing content. While both technologies have been a choice particularly made for this thesis, most of the concepts are also applicable to a wider range of technologies. The chapters 2 and 3 have been giving an basic and rather technology-independent introduction into the fields of enterprise applications and content applications. Most of the work done during this thesis is intended to be applicable to these areas in general.

The concept and descision process introduced in chapter 4 is fully independent from any concrete technology. Hence, the need for a hybrid persistence as well as the hybrid persistence itself can be transformed to every other technology introduced earlier.

The implementation instead is per se far more tightly coupled to specific platforms and technologies. To provide abstraction and generality to a certain degree, it was nevertheless introduced using three consecutive steps, each one getting more concrete than the previous one:

- **Conceptual level:** Section 5.4 introduced the basic aspects of the implementation. This is still abstract and independent from technological details and thus should be transformable to a high degree.

- **API:** Section 5.5 introduced the interfaces ContentManger, ContentConfiguration und Content. They are decoupling the individual parts and the application from each other component to a certain degree. This way, individual components, classes and ascpects can be exchanged by other implementations. For example the interceptor could easily be exchanged by another activation method, e.g. by making calls to the content manager programatically. By still using the conent manager interface this would not affect any other component.
- **Implementation:** Furthermore section 5.5 explained detailed implementational solutions. This is the part of the implementation that is specifically done for the actual target standards, Java EE and JCR.

Chapter 7

Conclusion

7.1 Future Work

Future work recommended to achieve on the first major topic, the decision process introduced in chapter 4, would have to contain:

- The decision process only takes data and content features into account. The third factor, the expected integration complexity, hasn't been taken into account. If this would be included in the process a complete cost estimation could be achieved.
- The methods used to analyze the existing data would have to be improved. On one hand this means minor fixes and improvements to make the methods better and the results more sigificant and comparable between different projects. But also especially for the data method and the semantic method support in development environments like Eclipse would be imaginable. But also for the survey method support would make execution of the survey more efficient. E.g. the required content-features could be collected as tagged information which a plugin could read and transform into a report. Also summarizing reports for whole projects would increase the performance.
- The methods, or the whole process respectively, could be worked into the existing processes ATAM and CBAM, introduced in section 4.2.1. This could help to popularize the decision process when it is part of a popular and widely accepted process.
- Finally the process would have to be applied to projects in order to gather results and experiences. This could on one hand lead to improvements on the process itself. On the other hand, it might also lead to a result base from which concrete numbers could be derived.

Future work in detail that is recommended to achieve on the second major topic, the framework CEF which was introduced in chapter 5, has already been covered in section 5.7. But from a rather high level it would definitely be desirable to do the following:

- First of all, CEF would have to be finished and made ready for usage in a production project. This requires to finish most of the points mentioned in section 5.7. Maybe some more improvements have to be done in order to customize it to a certain project situation.
- Experience has to be collected in real-world productive projects.

There was the assumption that hybrid systems, being specialised in processing content data, will under most conditions (regarding average number of content features required, an average integration complexity and the like) bring more benefit at lower costs than integrating content-related features into enterprise applications. But then again, one of the most popular O/R mapping frameworks for enterprise persistence, Hibernate, has recently integrated Apache Lucene.[1] This enables full-text search over Hibernate-persisted entity objects. One one hand, this shows the need for content-related features in enterprise applications. On the other hand, this area should be observed in the future, there might be an alternative solution coming up.

7.2 Summary

This thesis didn't coin the terms of the structuredness of data nor did it develop the hybrid implementation model that was introduced in chapter 4. All that has been around for some time. It rather linked a fundamental part of every software, the persistent data, together with a fundamental decision to be made in every project, the architecture. Even though the proposed architectural options are already in use all over the place, there has been little awareness about them and about how they are directly influenced by data types and by operations that might have to be performed on those data.

This thesis showed that the kind of data an application is required to process has direct influence on the appropriate model of implementation. It showed how this influence can be measured, documented and how it can be used well-regulatedly in order to determine the right architectural choices. It also helps to move this decision into an early stage of development. This helps to circumvent wrong architectural decisions that are realized too late during the project.

It further defined the term of the integration depth in conjunction with hybrid implementation models and explained the cost behaviour in comparison with the main alternative, a traditional, single-system enterprise application.

Based on that it implemented a framework based on Java EE 5 and JCR. Through its realized *total integration* approach it makes follow-up costs of hybrid projects independent from the integration depth. Furthermore it lowers the barrier for a hybrid implementation and reduces initial costs to a certain degree. In the end, it only requires very little effort to enable enterprise applications to use content-related features like full-text search, search over binaries or versioning. Even more, it opens the well-established world of Java enterprise applications up for the world of unstructured and semi-structured data, one more time facing an estimated 80 percent of all existing data within organizations are unstructured[2]. By seamlessly integrating both worlds, it paid attention to not re-inventing the wheel but providing enough glue code to let both application types play their own strengths and letting the other ones play theirs.

The rising appearance of the word "'unstructured"', rarely to be heard before by many people, emphasizes the rising importance of content. Ever-growing network speeds all over the world, dropping prices for storage devices, the success of multimedia producing masses of different outputs in various formats, just to mention a few. All those factors, and many more probably, lead to an explosion in content data. The amount of information existing all over the world grew from 3.000.000.000 (billion) gigabytes in 2000 to an estimated 24.000.000.000 gigabytes in 2003.[3] By the time of writing this is more than four years ago and the growth

[1] See *lucene.apache.org* and *www.hibernate.org*.

[2] According to [RK05].

[3] According to estimations made by UC Berkely in "'How much information?"'. The full report can be found under *http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/*.

rate is even expected to still rise. Hence, effective management and integration of content is becoming one of the most important topics over the next years.

In this context however, the proposed solution is not meant to be the best shot and a substitute to high-end full-fledged enterprise content management systems. The theoretical considerations about the role of data and data types should rather rise awareness when designing enterprise applications in general. It doesn't necessarily have to lead to a hybrid implementation model using JCR and Apache Jackrabbit. Even in larger scales, awareness about different categories of data might help to distribute them in a senseful way across different systems within an IT landscape. The principles behind the different types of data, the different types of applications as well as the cost drivers associated with them stay more or less the same in scenarios that are differing from the scenario investigated in this thesis. And last but not least, enterprise applications integrating content data from enterprise content management systems have to deal mostly with the same issues as the ones that have been solved with CEF.

Appendix A

Links

Name	URL
camunda GmbH	http://www.camunda.com
JSR-170 (JCR 1.0)	http://www.jcp.org?id=170
JSR-283 (JCR 2.0)	http://www.jcp.org?id=283
Apache Jackrabbit	http://jakrabbit.apache.org
JCR-Explorer	http://www.jcr-explorer.org
Architecture Tradeoff Analysis Method	sei.cmu.edu/architecture/ata_method.html
Cost Benefit Analysis Method	sei.cmu.edu/architecture/products_services/cbam.html
EMC Documentum	http://www.emc.com
Microsoft Sharepoint	http://office.microsoft.com/en-us/sharepointserver
Day Communiqué	http://www.day.com

Table A.1: Links

Appendix B

Sources

B.1 Interceptors

B.1.1 PersistInterceptor

Listing B.1: Source code PersistInterceptor

```
1  package com.camunda.research.jcr.cef.impl.ocm.ejb3interceptor;
2
3  public class PersistInterceptor {
4    ContentManager contentManager;
5
6    @AroundInvoke
7    public Object persist(InvocationContext invocation) throws Exception {
8      Object result = invocation.proceed();
9      if (contentManager == null) {
10       contentManager = new JackrabbitOcmContentManager();
11     }
12
13     for (Object param : invocation.getParameters()) {
14       if (AnnotationHelper.isContentEnriched(param)) {
15         contentManager.persist(param);
16       }
17     }
18
19     return result;
20   }
21 }
```

B.1.2 PersistCollectionInterceptor

Listing B.2: Source code PersistCollectionInterceptor

```
1  package com.camunda.research.jcr.cef.impl.ocm.ejb3interceptor;
2
3  public class PersistCollectionInterceptor {
4
5    @EJB(name = "ejb/JackrabbitOcmContentManager")
6    ContentManager contentManager;
7
8    @AroundInvoke
9    public Object persist(InvocationContext invocation) throws Exception {
10     Object result = invocation.proceed();
11
12     for (Object param : invocation.getParameters()) {
13       if (param instanceof Collection) {
14         contentManager.persistCollection((Collection<Object>) param);
15       }
```

```
16      }
17
18      return result;
19    }
20  }
```

B.1.3 LoadInterceptor

Listing B.3: Source code LoadInterceptor

```
1  package com.camunda.research.jcr.cef.impl.ocm.ejb3interceptor;
2
3  public class LoadInterceptor {
4
5    @EJB(name = "ejb/JackrabbitOcmContentManager")
6    ContentManager contentManager;
7
8    @AroundInvoke
9    public Object persist(InvocationContext invocation) throws Exception {
10     Object result = invocation.proceed();
11     result = contentManager.loadContent(result);
12     return result;
13   }
14 }
```

B.1.4 LoadCollectionInterceptor

Listing B.4: Source code LoadCollectionInterceptor

```
1  package com.camunda.research.jcr.cef.impl.ocm.ejb3interceptor;
2
3  public class LoadCollectionInterceptor {
4
5    @EJB(name = "ejb/JackrabbitOcmContentManager")
6    ContentManager contentManager;
7
8    @AroundInvoke
9    public Object persist(InvocationContext invocation) throws Exception {
10     Object result = invocation.proceed();
11     if (result instanceof Collection) {
12       result = contentManager
13         .loadContentCollection((Collection<Object>) result);
14     }
15     return result;
16   }
17 }
```

B.1.5 DeleteInterceptor

Listing B.5: Source code DeleteInterceptor

```
1  package com.camunda.research.jcr.cef.impl.ocm.ejb3interceptor;
2
3  public class DeleteInterceptor {
4
5    @EJB(name = "ejb/JackrabbitOcmContentManager")
6    ContentManager contentManager;
7
8    @AroundInvoke
9    public Object persist(InvocationContext invocation) throws Exception {
10     Object result = invocation.proceed();
11     Object[] params = invocation.getParameters();
```

```
12
13      if ((params == null) || (params.length < 2)) {
14        return null;
15      }
16      contentManager.remove((Class) params[0], params[1]);
17      return result;
18    }
19  }
```

B.2 Example Entity Object Enriched with Content

Listing B.6: Example Entity Object Enriched with Content

```
1  @Entity
2  @Table(name = "Customer")
3  @ContentEnriched
4  @ContentPathPattern("People/Customers")
5  @ContentClass(CustomerContent.class)
6  public class Customer implements Serializable {
7    private int id;
8
9    private String name;
10
11   @Id
12   @GeneratedValue(strategy = GenerationType.AUTO)
13   @ContentId
14   public int getId() {
15     return id;
16   }
17
18   public void setId(int id) {
19     this.id = id;
20   }
21
22   @Transient
23   private CustomerContent content;
24
25   @GetAssociatedContent
26   @Transient
27   public CustomerContent getContent() {
28     return content;
29   }
30
31
32   @SetAssociatedContent
33   @Transient
34   public void setContent(Object content) {
35     this.content = content;
36   }
37 }
```

B.3 Interface ContentConfiguration

Listing B.7: Interface ContentConfiguration

```
1  package com.camunda.research.jcr.cef.impl.ocm;
2
3  public interface ContentConfiguration {
4
5    public String getEntityRootPath(Object entity);
6
7    public String getEntityRootPath(Class entityClass, Object id);
8
```

```
9   public String getCollectionRootPath(Class entityClass);
10
11  public String getCollectionRootPath(Object entity);
12
13  public Class<Object> getContentClass(Object entity);
14
15  public String getId(Object entity);
16
17  public Object setContent(Object entity, Object content);
18
19  public Content getContent(Object entity);
20
21  public String getPathPattern(Class<Object> entityClass);
22
23  public boolean isContentEnriched(Object entity);
24
25  public String getRoot();
26
27 }
```

B.4 JackrabbitOcmContentManager

Listing B.8: Implementation of JackrabbitOcmContentManager

```
1  package com.camunda.research.jcr.cef.impl.ocm;
2
3  import com.camunda.research.jcr.cef.util.RepositoryHelper;
4
5  @Stateless
6  @Local(ContentManager.class)
7  public class JackrabbitOcmContentManager implements ContentManager {
8
9    private javax.jcr.Session session;
10
11   private org.apache.jackrabbit.ocm.manager.impl.
12                               ObjectContentManager objectContentManager;
13
14   private ContentConfiguration contentConfiguration;
15
16   public JackrabbitOcmContentManager() {
17     try {
18       InitialContext ctx = new InitialContext();
19       Repository repository = (Repository) ctx.lookup("java:jcr/local");
20       Credentials credentials = new SimpleCredentials("EJB-APP",
21                                                       "".toCharArray());
22       this.session = repository.login(credentials);
23
24
25       InputStream mapping = this.getClass().getResourceAsStream(
26                                                 "/cef-ocm-mapping.xml");
27
28       this.objectContentManager = new ObjectContentManagerImpl(session,
29                                       new InputStream[] { mapping });
30
31       this.contentConfiguration = new AnnotationContentConfiguration();
32     } catch (Exception e) {
33       throw new ContentStorageException(
34           "JCR not available, see nested exception for details.", e);
35     }
36   }
37
38   public Object load(Object entity) {
39     if (entity == null) {
40       return entity;
41     }
42
43     if (contentConfiguration.isContentEnriched(entity) == false) {
```

```
44      return entity;
45    }
46
47    Object c = loadEntity(entity);
48    entity = contentConfiguration.setContent(entity, c);
49
50    return entity;
51  }
52
53  private Object loadEntity(Object entity) {
54    return load(contentConfiguration.getContentClass(entity),
55      contentConfiguration.getCollectionRootPath(entity),
56      contentConfiguration.getId(entity));
57  }
58
59  private Object load(Class contentClass, String path, String id) {
60    Filter filter = objectContentManager.getQueryManager().
61                                              createFilter(contentClass);
62    filter.setScope(path + "//");
63    filter.addEqualTo("id", id);
64    Query query = objectContentManager.getQueryManager()
65                                                    .createQuery(filter);
66    Object c = objectContentManager.getObject(query);
67    return c;
68  }
69
70  private Object loadSingleContent(Class classname, String path) {
71    Filter filter = objectContentManager.getQueryManager().
72                                                createFilter(classname);
73    filter.setScope(path);
74    filter.addEqualTo("ocm:discriminator", classname.getCanonicalName());
75    Query query = objectContentManager.getQueryManager()
76                                                    .createQuery(filter);
77    Object c = objectContentManager.getObject(query);
78    return c;
79  }
80
81  public void persist(Object entity) {
82
83    if (entity == null) {
84      return;
85    }
86
87    if (contentConfiguration.isContentEnriched(entity) == false) {
88      return;
89    }
90
91    if (contentConfiguration.getContent(entity) == null) {
92      return;
93    }
94
95    try {
96      prepareContentStructure(entity);
97    } catch (RepositoryException e) {
98      throw new ContentPersistenceOperationFailedException(
99      "Content structure could not be prepared!", e);
100   }
101
102   persistContent(contentConfiguration.getContent(entity),
103     contentConfiguration.getId(entity).toString(),
104     contentConfiguration.getEntityRootPath(entity), entity
105                                       .getClass().getCanonicalName());
106 }
107
108 public void remove(Object entity) {
109   if (entity == null) {
110     return;
111   }
112
113   String id = contentConfiguration.getId(entity);
```

```
114     remove(entity.getClass(), id);
115   }
116
117   public void remove(Class entityClass, Object id) {
118     String path = contentConfiguration.getEntityRootPath(
119                                                         entityClass, id);
120
121     try {
122       if (session.itemExists(path)) {
123         session.getItem(path).remove();
124         session.save();
125       } else {
126         logger.warn("Content should be deleted but does not exist.");
127       }
128     } catch (Exception ex) {
129       logger.error("Error removing content.", ex);
130     }
131   }
132
133   private void prepareContentStructure(Object entity)
134       throws RepositoryException {
135     RepositoryHelper.createNode(session, contentConfiguration
136                                       .getCollectionRootPath(entity));
137   }
138
139   public Collection<Object> loadCollection(Collection<Object> entities) {
140     Collection result = new ArrayList();
141     for (Object o : entities) {
142       result.add(load(o));
143     }
144     return result;
145   }
146
147   private void persistContent(Content content, String id, String path,
148         String entityClassname) {
149
150     try {
151       content.setId(id);
152       content.setPath(path);
153
154       boolean checkedOut = false;
155       if (objectContentManager.objectExists(content.getPath())) {
156
157         try {
158           objectContentManager.checkout(content.getPath());
159           checkedOut = true;
160         } catch (VersionException e) {
161           logger.debug(content.getPath()
162                                + " is not versionable apparantly.");
163         }
164
165         logger.warn("Content " + content.getId()
166             + " already exists. Updating instead of inserting.");
167         objectContentManager.update(content);
168
169       } else {
170         objectContentManager.insert(content);
171
172         // On first persist operation for this entity set
173         // cef:entityClassname field
174         RepositoryHelper.setProperty(session, path,
175             "cef:entityClassname", entityClassname);
176
177       }
178
179       objectContentManager.save();
180
181       if (checkedOut) {
182         objectContentManager.checkin(content.getPath());
183       }
```

```
184
185     } catch (Exception ex) {
186       throw new ContentPersistenceOperationFailedException(
187         "Error on persisting content!", ex);
188     }
189   }
190
191   public void persistCollection(Collection<Object> entities) {
192     for (Object entity : entities) {
193       persist(entity);
194     }
195   }
196
197   public ContentConfiguration getContentConfiguration() {
198     return this.contentConfiguration;
199   }
200
201 }
```

B.5 Content Mapping Description

Listing B.9: Content Mapping Description (incomplete)

```
1  <?xml version="1.0" standalone="yes"?>
2
3  <jackrabbit-ocm>
4
5    <class-descriptor
6      className="c.c.r.jcr.ejb3.article.content.ProductContent"
7      jcrNodeType="nt:unstructured" discriminator="true"
8      jcrMixinTypes="cef:businessEntity,  mix:versionable">
9      <field-descriptor fieldName="path" path="true" />
10     <field-descriptor fieldName="id" jcrName="id" id="true" />
11     <field-descriptor fieldName="shortDescription" jcrName="shortDescription" />
12     <field-descriptor fieldName="longDescription" jcrName="longDescription" />
13     <field-descriptor fieldName="technicalData" jcrName="technicalData" />
14     <field-descriptor fieldName="scopeOfDelivery" jcrName="scopeOfDelivery" />
15     <field-descriptor fieldName="manufacturer" jcrName="manufacturer" />
16     <bean-descriptor fieldName="dataSheet" jcrName="dataSheet" />
17     <bean-descriptor fieldName="images" jcrName="images" />
18   </class-descriptor>
19
20   <class-descriptor
21     className="com.camunda.research.jcr.cef.impl.ocm.common.BinaryDocument"
22     jcrNodeType="nt:resource">
23     <field-descriptor fieldName="data" jcrName="jcr:data" />
24     <field-descriptor fieldName="mimeType" jcrName="jcr:mimeType" />
25     <field-descriptor fieldName="lastModified" jcrName="jcr:lastModified" />
26     <field-descriptor fieldName="encoding" jcrName="jcr:encoding" />
27   </class-descriptor>
28
29 </jackrabbit-ocm>
```

Bibliography

[Abi97] Serge Abiteboul. Querying semi-structured data. January 1997.

[AH00] Gunter Saake Andreas Heuer. *Datenbanken: Konzepte und Sprachen*. MITP-Verlag, 2000.

[AK04] André Eickler Alfons Kemper. *Datenbanksysteme*. Oldenbourg Wissenschaftsverlag GmbH, 2004.

[All] Carl Allen. Software maintenance - an overview. http://www.bcs.org/server.php?show=ConWebDoc.3063. Visited on January 13th 2008.

[BDKZ93] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. Software complexity and maintenance costs. *Commun. ACM*, 36(11):81–94, 1993.

[Boe81] Barry Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[Cod90] E. F. Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Dij82] Edsger W. Dijkstra. On the role of scientific thought. *Selected writings on Computing: A Personal Perspective*, 1982.

[DK75] Frank DeRemer and Hans Kron. Programming-in-the large versus programming-in-the-small. In *Proceedings of the international conference on Reliable software*, pages 114–121, New York, NY, USA, 1975. ACM.

[Fie05] Roy T. Fielding. Jsr 170 overview. March 2005.

[GH03] Bobby Woolf Gregor Hohpe. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman, 2003.

[JA] Rick Kazman Jai Asundi. A foundation for the economic analysis of software architectures.

[JA01] Mark Klein Jai Asundi, Rick Kazman. Using economic considerations to choose among architecture design alternatives. 2001.

[(JC01] Java Community Process (JCP). Java specification request 143: Workspace versioning and configuration management. http://www.jcp.org/en/jsr/detail?id=147, January 2001. Visited on September 18th 2007.

[(JC02] Java Community Process (JCP). Jsr 907: Java transaction api (jta). http://www.jcp.org/en/jsr/detail?id=907, November 2002. Visited on January 11th 2008.

[(JC05a] Java Community Process (JCP). Jsr 170: Content repository for java technology api. http://www.jcp.org/en/jsr/detail?id=170, June 2005. Visited on January 10th 2008.

[(JC05b] Java Community Process (JCP). Jsr 283: Content repository for java technology api version 2.0. http://www.jcp.org/en/jsr/detail?id=283, September 2005. Visited on January 11th 2008.

[(JC06a] Java Community Process (JCP). Java specification request 220: Enterprise javabeans 3.0. http://www.jcp.org/en/jsr/detail?id=220, May 2006. Visited on December 6th 2007.

[(JC06b] Java Community Process (JCP). Java specification request 244: Java platform, enterprise edition 5 (java ee 5) specification. http://www.jcp.org/en/jsr/detail?id=244, May 2006. Visited on January 11th 2008.

[JG93] Andreas Reuter Jim Gray. *Transaction Processing: Concepts and Techniques*. 1993.

[KA89] P. Kanellakis and S. Abiteboul. Database theory column. *SIGACT News*, 20(4):17–23, 1989.

[Lad03] Ramnivas Laddad. *AspectJ in Action*. Manning, 2003.

[Lib05] Jesse Liberty. *Programming C#: Building .NET Applications with C#*. O'Reilly, 19812005.

[MB07] Bernd Rücker Martin Backschat. *Enterprise Java Beans 3.0*. Elsevier, Spektrum Akademischer Verlag, 2007.

[MF02] Matthew Foemmel Martin Fowler, David Rice. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman, 2002.

[Mica] Sun Microsystems. Annotations. http://java.sun.com/j2se/1.5.0/docs/guide/language/annotati
Visited on January 6th 2008.

[Micb] Sun Microsystems. Api specification: Java transaction api (jta). http://java.sun.com/products/jta/index.html. Visited on September 18th 2007.

[PAB97] Eric Newcomer Phillip A. Bernstein. *Transaction Processing: For the Systems Professional*. 1997.

[PB97] Mary Fernandez Dan Suciu Peter Buneman, Susan Davidson. Adding structure to unstructured data. January 1997.

[RE07] Shamkant B. Navathe Ramez Elmasri. *Fundamentals of Database Systems*. Addison-Wesley, 2007.

[RK01] Mark Klein Rick Kazman, Jai Asundi. Quantifying the costs and benefits of architectural decisions. 2001.

[RK05] A. White Rita Knox, T. Eid. Management update: Companies should align their structured and unstructured data. February 2005.

[SA00] Dan Suciu Serge Abiteboul, Peter Bunemann. *Data on the Web*. Morgan Kaufmann, 2000.

[Sne95] Harry N. Sneed. Estimating the costs of software maintenance tasks. 1995.

[Ull07] Christian Ullenboom. *Java ist auch eine Insel - Das umfassende Handbuch*. Galileo Computing, 2007.

[Unia] Carnegie Mellon University. The architecture tradeoff analysis method (atam). http://www.sei.cmu.edu/architecture/ata_method.html. Visited on Dezember 20th 2007.

[Unib] Carnegie Mellon University. Cost benefit analysis method (cbam). http://www.sei.cmu.edu/architecture/products_services/cbam.html. Visited on Dezember 20th 2007.

List of Figures

List of Tables