

Institute of Architecture of Application Systems  
University of Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Bachelor Thesis No. 171

## **Integration and Extension of a Cloud Data Migration Support Tool**

Andreas Rempel



**Course of Study:** Software Engineering

**Examiner:** Prof. Dr. Dr. h. c. Frank Leymann

**Supervisors:** Steve Strauch

Dr. Vasilios Andrikopoulos

**Commenced:** September 18, 2014

**Completed:** May 20, 2016

**CR-Classification:** C.2.4, D.2.5, D.2.7, D.2.12, H.2.4



## Abstract

Since the growth of Cloud computing, the desire to use this novel computing approach has increased. It is not necessary to redevelop an existing application to target the Cloud and benefit from its advantages. Sometimes, it is even more sensible to migrate existing applications running in a static environment. Since many of those application have strict layers, where not each layer might benefit from an elastic hosting environment, it is sometimes sufficient to migrate only individual layers. To cover as many use cases as possible, broad convertible scenarios, not uncommonly going beyond proprietary approaches, must be offered. Because a migration process is still a pretty complex matter, it is convenient to have a guided conversion to cover all requirements and achieve the desirable result.

This bachelor thesis focuses on aspects of migrating data to the Cloud by using a previously developed prototype of a Cloud Data Migration Support Tool. Particularly, the integration of already existing modifications and evaluations of this tool, which were already developed independently, into one stable prototype were required. A further objective of this thesis is to gain platform independence by extending the prototype by a plug-in mechanism to allow the use of native *Java DataBase Connectivity* (JDBC) drivers for exporting data from existing storage sources and subsequently importing this data into a target data environment, whose types may differ from the types of the source data environment. Furthermore, applying proven concepts on architecture and design are part of this work as well.



---

# Contents

---

<b>1. Introduction</b>	<b>1</b>
1.1. Motivating Scenario . . . . .	2
1.2. Problem Statement and Scope . . . . .	3
1.3. Definitions and Conventions . . . . .	4
1.3.1. Terms . . . . .	4
1.3.2. Abbreviations . . . . .	4
1.4. Outline . . . . .	6
<b>2. Fundamentals</b>	<b>9</b>
2.1. Cloud Computing . . . . .	9
2.2. Differences of Databases . . . . .	11
2.2.1. Relational Databases . . . . .	11
2.2.2. Object Relational Databases . . . . .	12
2.2.3. Column-Family Stores . . . . .	13
2.2.4. Document Databases . . . . .	14
2.2.5. Key-Value Stores . . . . .	15
2.2.6. Graph Databases . . . . .	16
2.3. Data Migration . . . . .	18
2.3.1. Challenge . . . . .	19
2.3.2. Migration Tool and Methodology - Bachmann . . . . .	19
2.4. Software Integration Strategies . . . . .	22
2.4.1. Bottom-up Integration . . . . .	22
2.4.2. Top Down Integration . . . . .	25
2.4.3. Conclusion . . . . .	26
2.5. Java Extension Mechanism . . . . .	27
2.5.1. Class Loading . . . . .	27
2.5.2. Extension Loading . . . . .	28
<b>3. Related Work</b>	<b>31</b>
3.1. Previous Results . . . . .	31
3.1.1. Migration Scenario: RDBMS to NoSQL . . . . .	31
3.1.2. Evaluation of Methodology for DBL Migration: Industry . . . . .	32
3.1.3. Evaluation of Methodology for DBL Migration: eScience . . . . .	34
3.1.4. Positioning and Distinguishes . . . . .	35
3.2. Multi-tenant Open-Source Enterprise Service Bus . . . . .	35
<b>4. Concept and Specification</b>	<b>37</b>
4.1. Integration and Adaption . . . . .	37
4.1.1. Adapters . . . . .	38

4.1.2.	Functional Requirements . . . . .	38
4.2.	Add-on Extension Mechanism . . . . .	41
4.2.1.	Add Adapter . . . . .	42
4.2.2.	Delete Adapter . . . . .	43
4.2.3.	Assign Adapter . . . . .	44
4.2.4.	Dismiss Adapter . . . . .	45
4.3.	Non-Functional Requirements . . . . .	46
4.3.1.	Extensability . . . . .	46
4.3.2.	Usability . . . . .	46
4.3.3.	Reuseability . . . . .	46
4.3.4.	Integratability . . . . .	46
4.3.5.	Maintainability . . . . .	46
4.3.6.	Backward Compatibility . . . . .	47
4.3.7.	Security . . . . .	47
4.3.8.	Portability . . . . .	47
<b>5.</b>	<b>Design</b>	<b>49</b>
5.1.	Adaptations . . . . .	49
5.1.1.	Presentation Layer . . . . .	49
5.1.2.	Workflow . . . . .	50
5.2.	Extension Mechanism Architectural Overview . . . . .	53
5.2.1.	Architecture of the Extension Mechanism . . . . .	53
5.2.2.	DB Schema Extension . . . . .	55
<b>6.</b>	<b>Implementation</b>	<b>57</b>
6.1.	Implementation of the Integration . . . . .	57
6.1.1.	Tools and Libraries Used for Integration . . . . .	57
6.1.2.	Integration Strategy . . . . .	59
6.2.	Migration and Reconstruction . . . . .	60
6.2.1.	Motivation . . . . .	61
6.2.2.	Extended Toolset . . . . .	61
6.2.3.	Build Cycle . . . . .	62
6.2.4.	Deployment . . . . .	63
6.3.	Adaptation of Migrated Project . . . . .	64
6.3.1.	Libraries . . . . .	64
6.3.2.	User Control . . . . .	65
6.4.	Extension of CDMT . . . . .	66
6.4.1.	Web-UI Navigation Controls . . . . .	67
6.4.2.	Dashboard . . . . .	68
6.4.3.	Adapter Overview Page . . . . .	70
6.4.4.	New Adapter Page . . . . .	71
6.4.5.	Approval of Implementation and Integration . . . . .	74
6.5.	Exemplary Adapters . . . . .	74
6.5.1.	MySQL Source Adapter . . . . .	75
6.5.2.	PostgreSQL Target Adapter . . . . .	75

<b>7. Validation</b>	<b>77</b>
7.1. Tool and Services . . . . .	77
7.2. Test Data . . . . .	78
7.3. Test Cases . . . . .	80
7.3.1. Use Case Based . . . . .	80
7.3.2. Based on Previous Results . . . . .	80
7.4. Validation Result . . . . .	81
7.4.1. Outcome of Use Case Based Tests . . . . .	81
7.4.2. Outcome of Tests Based on Previous Results . . . . .	82
<b>8. Conclusion and Future Work</b>	<b>83</b>
<b>A. Interfaces</b>	<b>85</b>
A.1. Source Adapter Interface . . . . .	85
A.2. Target Adapter Interface . . . . .	86
<b>Bibliography</b>	<b>89</b>





---

## List of Figures

---

2.1. Relational Database Schema . . . . .	12
2.2. ORDB Data Model . . . . .	13
2.3. Cassandra Data Model . . . . .	14
2.4. MongoDB Data Model . . . . .	15
2.5. Key-Value Store Data Model . . . . .	16
2.6. Graph Data Model . . . . .	17
2.7. System Component Diagram and Layer Diagram of CDMT . . . . .	20
2.8. Methodology for Data Migration . . . . .	21
2.9. Isolated Modules Bottom-up Test . . . . .	23
2.10. Isolated Components Bottom-up Test . . . . .	23
2.11. Overall System Test . . . . .	24
2.12. Testing Highest Invoking Module . . . . .	25
2.13. Testing Components' Highest Invoking Module . . . . .	26
2.14. JRE Class Loader Hierarchy . . . . .	27
2.15. Schematic Functionality of a Class Loader . . . . .	28
3.1. Extended Methodology for the Migration Scenario RDBMS to NoSQL . . . . .	32
3.2. Methodology of AWS and Bachmann Combined . . . . .	34
3.3. Architecture of an ESB Instance . . . . .	36
4.1. Use Cases Overview . . . . .	41
5.1. Component Connection of the Presentation Layer . . . . .	50
5.2. Workflow for the Decision Support . . . . .	52
5.3. Add-on Mechanism Overview . . . . .	53
5.4. Snipped of Extended DB Schema . . . . .	55
6.1. Structure of Deployed System . . . . .	63
6.2. Initiative View After Creating a new Project . . . . .	65
6.3. Message Exchange Schema . . . . .	66
6.4. Welcome Page . . . . .	67
6.5. Navigation Concept of CDMT . . . . .	68
6.6. Dashboard of CDMT . . . . .	69
6.7. Adapters Overview . . . . .	70
6.8. Available Actions . . . . .	71
6.9. Page for Adding a new Adapter . . . . .	72
6.10. File Upload Area in Detail . . . . .	73
7.1. Crowfoot Diagram of the Test DB . . . . .	79



---

## List of Tables

---

2.1. Overview of DB Types . . . . .	19
2.2. Data Migration Scenarios by Bachmann . . . . .	22
3.1. First Iteration of Evaluation by Nachev . . . . .	33
3.2. Second Iteration of Evaluation by Nachev . . . . .	33
3.3. Envaluation Iterations by Guo . . . . .	34
4.1. Overview of Provided Adapters . . . . .	38
4.2. Functional Requirements for CDMT After Integration . . . . .	40
4.3. Use Case Add Adapter . . . . .	42
4.4. Use Case Delete Adapter . . . . .	43
4.5. Use Case Assign Adapter . . . . .	44
4.6. Use Case Dismiss Adapter . . . . .	45
7.1. Test Case Migration . . . . .	81



---

## List of Listings

---

2.1. PostgreSQL Inheritance Example . . . . .	13
2.2. Neo4J Indexed Search Example . . . . .	18
2.3. Fragment of Interface Definition . . . . .	28
2.4. Fragment of Add-on Implementation . . . . .	29
2.5. Extend Class Loader Snippet . . . . .	29
2.6. Snippet to Load a Plug-in . . . . .	29
A.1. Source System Interface . . . . .	85
A.2. Target System Interface . . . . .	86



---

# 1. Introduction

---

With its pay per use approach Cloud computing changed not only business model of vendors in the area of IT, but rather the way of application development and data management.

A considerable number of business branches have to deal with fluctuation in the utilisation of their systems, since some tasks are done at equal intervals and mostly concurrently. Between the peaks of workloads a recognizable part of their infrastructure runs at idle, hence the arising costs for running the entire infrastructure without gaining any profit.

Related problems arise for companies with an unpredictable growth using conventional IT solutions. On the one hand these companies have to face malinvestments by extending their systems too soon and before it is necessary, on the other hand a loss of revenue may occur if they wait too long and cannot handle the growing workload.

As a reimplementations of released software for a completely different and quite novel environments is prohibitive the fewest small and middle range software companies can afford it. A more practicable strategy is to adapt applications for the Cloud environment, as reasoned in [ABLS13]. Since many applications have layered architectures it is conceivable to migrate software of that kind layer-by-layer, or just the parts which profit from an elastic environment the most. To give an entry point for manage such an intention a Web-based *Cloud Data Migration Tool* (CDMT) was developed by Bachmann as part of his diploma thesis [Bac12] and further used for [SAK<sup>+</sup>14] and [SAB<sup>+</sup>13]. It supports many scenarios to migrate data between different environments and *databases* (DB) as well. Additionally it supports the user to find the most suitable scenario and gives advices for essential customisation of the application which data layer is to migrate. The above-mentioned tool was evaluated and extended in the master's thesis of Lamllari [Lam13] as well as the diploma thesis of Nachev [Nac13], which will be used as basis for this thesis.

## 1.1. Motivating Scenario

In the past decades the Internet and technologies associate to it became more prevailing, which was especially amplified by the upcoming of Web 2.0. Many customers as well as vendors tending to use and respectively provide off-premise solutions for IT products. These trend is encouraged, among other things, by the properties of Cloud computing. In respect to provide services it is of high importance to support software architectures which leverage the exchangeability of function or service providing components. However, using services of third party providers is a common approach among software companies. If we consider the mentioned Web-based CDMT, on the one hand it supports anyone interested to provide individual applications for deployment in a Cloud environment, as introduced in [SAB<sup>+</sup>13], but on the other hand it remains an isolated and slightly scalable application itself. Furthermore, to extend CDMT changes have to be done within its source code. Since specifications of *Cloud Data Hosting Solutions* (CDHS) can be changed by the vendors to fulfil requirements that arise with new technologies. Another side effect of progress in the context of Cloud computing is the appearance of new vendors. To prevent CDMT getting obsolete is an appropriate consideration to extend its architecture to enable an extension mechanism that does not claim to access the original source code. For that approach many software products, like Mozilla Firefox<sup>1</sup>, have implemented an "add-on functionality" to support extensions that fulfil the desires of their users. In addition, not only the desires of users can be satisfied by such an approach, but also changing requirements of the environment of a software to a certain extend.

A further aspect to take into account are *Not Only SQL* (NoSQL<sup>2</sup>) [SF12] databases. Since their upcoming the maturity and diversity of this movement is growing successively. Additionally many of the support *out-scaling* which in turn unburdens data distribution and replication. In this case even the vendors services could profit from the usage of DBs in this vein, since it eases to use technologies like virtualization and clustering for hosting Cloud services, what again lowers the cost for hardware commodity and bypass the physical limit of up-scaling to a certain degree. Yet another plus enabled by this strategy is the exclusion of the DB as a single point of failure, since replication becomes easier and less costly compared to the same approach realized with the most *Relational DataBase Management System(s)* (RDBMS). Even big companies, which are also used on a daily bases - eBay<sup>3</sup>, Netflix<sup>4</sup>, just to mention a few- [Vai13] are utilizing products of such kind. Moreover, the potential of NoSQL databases was recognized by popular companies, like Oracle<sup>5</sup> to give one sample, providing their own NoSQL databases.

---

<sup>1</sup><https://www.mozilla.org/en-GB>

<sup>2</sup><http://nosql-database.org>

<sup>3</sup><http://http://www.ebay.com>

<sup>4</sup><https://www.netflix.com>

<sup>5</sup><http://www.oracle.com>



## 1.2. Problem Statement and Scope

The problem on the current state of the *CDMT* is that there are several different versions of it. Each has its own benefits but also some shortcomings. While the initiative version, which was implemented by Bachmann [Bac12], and the version came up by Nachev, only support proprietary NoSQL databases Lamllari extended the prototype by adaptors for *MongoDB*<sup>6</sup> and *Cassandra*<sup>7</sup> allowing using both of them as the target environment for data migration. Allowing a more differentiated migration of data, and provides additional use cases for *CDMT*, to satisfy requirements of even more users. Then again Nachev's results [Nac13] fix some issues with respect to usability and functionality of *CDMT* besides the direct context of data migration.

To increase the applicability of the introduced tool we will use the results of the work above outlined to benefit from the findings acquired during their elaboration. More precisely the existing prototypes of the tool will be integrated into one version which shall comprise already usable functions of each.

The common problem as yet of all the implementations, which we mentioned above, is that external tools are used to establish connection to RDBMS, such as *MySQL*<sup>8</sup> or *PostgreSQL*<sup>9</sup>, which provide the required drivers.

Proprietary software always brings limitations by conventions inflicted by vendors of the particular products. A far-reaching restriction, even if it is the only one, is the mutual compatibility of modules used as part of an application, which also restricts the extensibility and environmental independence of the respective software. To reduce the restrictions in respect of the extensibility, in the context of the connectivity to *MySQL* and *PostgreSQL*, we will implement an add-on mechanism.

Furthermore, we will provide exemplary adapters, which use *JDBC*<sup>10</sup> to establish connection to a respective store. Additionally, *JDBC*-based adapters are able to handle transactions with a store as well. One practical intention beneath that add-on mechanism is to even the way for the usage of *ESB*<sup>MT11</sup>, which was introduced in [SAGSL13] and [SALM12], and is using *JDBC* for database connectivity as well.

---

<sup>6</sup><https://www.mongodb.org>

<sup>7</sup><http://www.cassandra.apache.org>

<sup>8</sup><http://www.mysql.com>

<sup>9</sup><http://www.postgresql.org>

<sup>10</sup><http://www.oracle.com/technetwork/java/javase/jdbc/index.html>

<sup>11</sup><http://www.iaas.uni-stuttgart.de/esbmt>

### 1.3. Definitions and Conventions

Terms, abbreviation, and the meaning of each are defined in this section. This thesis uses these definitions correspondingly.

#### 1.3.1. Terms

<b>Add-on:</b>	A software component that adds functionality to a system without claiming to change the system's implementation.
<b>Component:</b>	A piece of software which couples modules of similar type. A component is enclosed and should only communicate with its environment through its defined interfaces.
<b>Horizontal scaling:</b>	Increasing storage, computing power, or connectivity by upgrading a particular machine, which may represent a node in a cluster or comparable structures.
<b>Module:</b>	Represents a collection of functions and interfaces, which are necessary to provide a fundamental functionality or service. An Example file handling.
<b>Plug-in:</b>	A synonym for Add-on
<b>Service:</b>	A set of programming interfaces and classes providing access to some specific application feature or functionality.
<b>Use Case:</b>	A sequence of interactions between an actor (or actors) and a system triggered by a specific actor, which produces a result for an actor [LL10].
<b>Vertical scaling:</b>	Adding more computing nodes in a network for the same purpose.
<b>Software life cycle:</b>	

#### 1.3.2. Abbreviations

<b>BLL:</b>	Business Logic Layer
<b>BS:</b>	Bootstrap
<b>CAP:</b>	Consistency, Availability, Partition tolerance
<b>CQL:</b>	Cassandra Query Language
<b>CDHS:</b>	Cloud Data Hosting Solution
<b>CDMT:</b>	Cloud Data Migration Tool

### 1.3. Definitions and Conventions

---

<b>CSI:</b>	Continual Service Improvement
<b>CSS:</b>	Cascading Style Sheets
<b>DAL:</b>	Data Access Layer
<b>DB:</b>	DataBase
<b>DBL:</b>	DataBase Layer
<b>DBMS:</b>	DataBase Management System
<b>DOM:</b>	Domain Object Language
<b>ESB:</b>	Enterprise Service Bus
<b>FK:</b>	Foreign Key
<b>FMC:</b>	Fundamental Modeling Concepts
<b>GUI:</b>	Graphical User Interface
<b>IDE:</b>	Integrated Development Environment
<b>ITIL:</b>	Information Technology Infrastructure Library
<b>JB:</b>	Java Business Integration
<b>JDBC:</b>	Java DataBase Connectivity
<b>JRE:</b>	Java Runtime Engine
<b>JSP:</b>	Java Server Page
<b>JVM:</b>	Java Virtual Machine
<b>IaaS:</b>	Infrastructure as a Service
<b>NoSQL:</b>	Not only Structured Query Language
<b>PaaS:</b>	Platform as a Service
<b>PK:</b>	Primary Key
<b>POM:</b>	Project Object Language
<b>RAM</b>	Random Access Memory
<b>RDBMS:</b>	Relational DataBase Management System
<b>SaaS:</b>	Software as a Service
<b>SOA:</b>	Service-Oriented Architecture
<b>SQL:</b>	Structured Query Language
<b>UI:</b>	User Interface
<b>UML:</b>	Unified Modeling Language
<b>UUID:</b>	Universally Unique Identifier
<b>VM:</b>	Virtual Machine
<b>VS</b>	Visual Studio

## 1.4. Outline

This section gives a short overview of the following chapters with a brief description of their content. Each chapter covers a different aspect and stage of work. Almost all chapters are divided in sections, which again are subdivided in subsections to give a specific and precise scope and to focus on every certain issue separately.

- **Fundamentals (Chapter 2):**

This chapter lists and explains basic theoretical approaches used in this thesis. At the beginning it contains the main properties of Cloud Computing (Section 2.1), followed by Differences of Databases (Section 2.2) introducing NoSQL with focus on Cassandra and MongoDB including their mutual distinctions. Additionally we introduce Graph databases and key-value-stores to complete the list of the most common stores of the NoSQL domain. The third subsection handles basics of Data Migration (Section 2.3). Subsequent Section (see Sect. 2.4) deals with Integration Strategies and depicts the two most common ones. This chapter finishes with Section 2.5 in which we introduce the Java extension mechanism and give a short example of its usage.

- **Related Work (Chapter 3):** In this chapter we encompass previous work which is related the context of this thesis. Beginning with Section 3.1 we give an overview of recently published work which focus on the CDMT with different emphasis. In its Subsection 3.1.1 we present findings of Lamllari elaborated in [Lam13] by extending the methodology of the CDMT. Afterwards Subsection 3.1.2 gives an insight in Nachev's work of an evaluation of the CDMT by migrating a *Database Layer* (DBL) in cooperation with a company [Nac13]. The last work in this scope we present in Subsection 3.1.3. It introduces an evaluation of the CDMT as well, but was done by Guo in a scientific context as part of his thesis [Guo13]. We terminate this section by positioning our work compared to the previously introduced in Subsection 3.1.4.

In Section 3.2 we present concepts of ESB<sup>MT</sup>, introduced by Strauch et al. [SAGSL13], which are important for this thesis.

- **Specification (Chapter 4):** This chapter scopes requirements towards an extension of CDMT prototype's functionality. In Section 4.1, we outline functionally features and topics, which have to be fulfilled after integrating different versions of CDMT. In Section 4.2 we specify functional requirements for the extension of CDMT, besides of previous realisations and suggestions. Finally, we present non-functional requirements which are valid throughout all programmatic tasks, in Section 4.3 .

- **Design (Chapter 5):** This chapter ties on the specification and explains the design, which we use, to fulfil requirements, that are imposed in Chapter 4. In Section 5.1, we explain how the adaptations of the presentation layer and the workflow lead to fulfilment of the specified requirements of the integration. In Section 5.2 we give an overview of the system's architecture. Furthermore, it contains several extensions to provide a higher functionality of the CDMT.

- **Implementation (Chapter 6)** In this chapter we present the impact on CDMT by implementing the design and architectural changes. Section 6.1 has its scope on the implementation of the integration. In Section 6.2, we introduce the innovative development environment, tools, which we use for further development, as well as a defined build cycle, and a changed deployment. Afterwards, in Section 6.3, we introduce adaptations of the prototype, which provide a new way of interaction during migration projects. In Section 6.4 we describe the mechanism to load add-ons during runtime. Furthermore, we explain the modern navigation concept and Web-UI design, which we implement. Additionally, we outline what needs to be done to implement connectivity adapters as add-ons, in Section 6.5.
- **Validation (Chapter 7)** In this chapter, we describe the validation of our implementation and its outcome. In Section 7.1 we introduce tools and services, which we use for validation. Afterwards, we describe the test data we use, in Section 7.2. In Section 7.3, we explain which test cases we used to prove the correctness of our implementation, as well as how we defined them. Following this, we present the results of the validation, in Section 7.4.
- **Outcome (Chapter 8)** This is the final chapter. It summarizes our contribution to the subject of this work. Additionally, we provide suggestions on further improvement of CDMT and possible related topics for future work.



---

## 2. Fundamentals

---

This chapter narrows the domain and conceptual attempts to solve the given tasks. These attempts are used as fundamental to elaborate the solution. Furthermore, it aims at giving a better comprehension of the work which is done within the scope of this thesis. In Section 2.1, we explain the major principals of Cloud computing. We give a brief overview of distinctions between relational databases and different types of NoSQL data stores, such as document, or key-value stores for instance, in Section 2.2. We briefly describe the challenges of data migration in Section 2.3, as it is a topic of this work as well. Additionally this section gives a slight overview on the tool and methodology of Bachmann [Bac12], of which both are essential for this work. In Section 2.4, we explain the most common strategies for software integration as well as the assets and drawbacks may occur from using one of them. Finally, in Section 2.5, we introduce a mechanism to provide add-on functionality for Java applications.

### 2.1. Cloud Computing

In this section we give an introduction on Cloud computing. To clear what is meant in this thesis by the term *cloud computing* we provide an overview of the main aspects which define cloud computing.

First of all, we deliver a definition of *Essential Characteristics* published by *National Institute of Standards and Technology* (NIST) [MG11]:

- **On-demand self-service:** *"A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider."*
- **Broad network access:** *"Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations)."*
- **Resource pooling:** *"The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth."*
- **Rapid elasticity:** *"Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer,*

*the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time."*

- **Measured service:** *"Cloud systems automatically control and optimize resource use by leveraging a metering capability<sup>1</sup> at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service."*

Even large suppliers provide Cloud solutions for small business with the option of extending their infrastructure as soon as necessary. Furthermore, not every client wants or needs to set up every specific detail to provide his software in a remotely accessed manner. Some of these customers even pass on developing proprietary software solutions by reason of cost, time, or capabilities. Instead they rather would like to use a service providing needed functionality as long as they want to use it. To satisfy customers' desires, regarding the scope and type of IT systems, they can choose from three main *Service Models* [MG11]:

- **Software as a Service (SaaS):** Customers have their individual business models they want to implement. Vendors provide applications supporting customers' intentions.
- **Platform as a Service (PaaS):** Customers want to host applications in a certain environment. Vendors provide such environments for running individual applications, but also some mandatory criteria, like a certain style for developed applications.
- **Infrastructure as a Service (IaaS):** Customers are share common physical and virtual hardware. Vendors are managing this hardware.

Besides the above-mentioned decisions can be made a further topic is to chose a deployment model. The choice of a certain model often depends on security concerns, diversity, and extent of work shall be handled within the environment. The following four main *Deployment Models* [FLR<sup>+</sup>14] can be strictly distinguished:

- **Private Cloud:** IT resources are provided as a service exclusively to one customer in order to meet high requirements on privacy, security, and trust while enabling elastic use of static resource pool as good as possible.
- **Community Cloud:** IT resources are provided as a service to a group of customers trusting each other to enable collaborative elastic use of a static resource pool.
- **Public Cloud:** IT resources are shared among a huge customer group for the purpose of elastic use of a static resource pool.
- **Hybrid Cloud:** Different Cloud and static data centres are integrated to form a homogeneous hosting environment.

In addition each of these can be divided into more specific variants depending on the physical location of data inside the Cloud and the hardware used for the Cloud environment.



## 2.2. Differences of Databases

This section clears the main differences and challenge of migration from *Relational Databases* (RDB)s to DBs of other types. The following subsections discuss the main characteristics of the DBs which will be used in further steps of this thesis. As RDBs have just slight differences in usage and have a common model we just introduce their characteristics in general in Section 2.2.1. In Section 2.2.2 we introduce *Object Relational Databases* which have plenty properties in common with RDBs, but have the characteristic of inheritance as well. In Section 2.2.3 we discuss the way these *Column-Family Stores* work with focus on *CassandraDB*, since this is the DB we use. Section 2.2.4 depicts the functionality of *Document Databases* with respect to *MongoDB*. Short introductions of *Key-Value Stores* in Section 2.2.5 and *Graph Databases* in Section 2.2.6 will accomplish the listing of the currently most important NoSQL store types. Though the last two won't be used as part of this thesis they are important for the sake of completeness.

### 2.2.1. Relational Databases

The approach of *Relational Databases* was firstly discussed by E.F Codd in [Cod70]. Nowadays there are huge offering of RDBMS of which we use *MySQL* for this work.

A great benefit of *Structured Query Language* (SQL) based DB is that their query language is based on the *ISO/IEC 9075 standard*. Still not all vendors provide the same dialect yet and "*the number of differences that do exist can be confusing*"[O'N14]. Nevertheless it is not a hard task to understand the core of the other dialects when you already learned one [Far13]. This DB basically consist of tables which have mostly several columns. One column of each table should contain a *Primary Key* (PK), which is unique for each tuple of the particular table, to identify datasets. The rows of each table is homogeneous and can not vary in types of each cell or the amount of contained cells. A tuple is basically a row of a certain table. Every column in tables of a DB has a domain, which restricts the possible values which can be entered. A further characteristic in these is the *Database schema* (DBS). It has to be declared at the beginning and defines relations between tables, contained in the DB, and their structure, as well as the domains of each column, as depict in Figure 2.1.

Once a DBS is defined and used over a certain time it becomes a "*segnificant undertaking*" [VMZ<sup>+</sup>10] to alter it.

Relations between tables are represented by *foreign keys* whose values are identical to the values of PK in the table to link to. A down side of RDBs is the effort if you need to meet new requirements, you cannot make ad-hock changes, but have to adjust the schema. However, the effort for this task depends on the complexity of the changes.

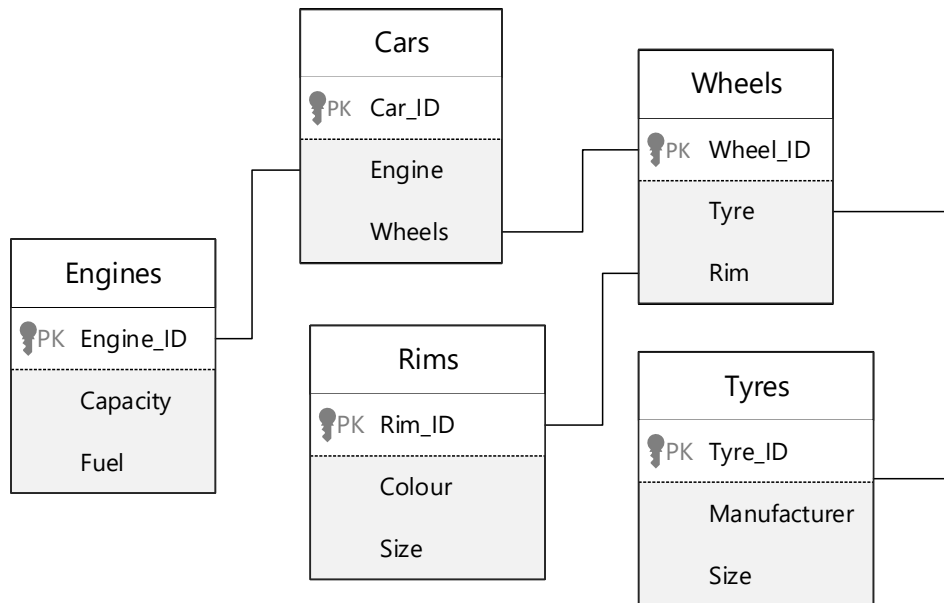


Figure 2.1.: Relational Data Model

### 2.2.2. Object Relational Databases

The content in this section is based on the official documentation of *PostgreSQL* [Gro09]. Object relational databases store data, like their relatives RDBs described in previous section, in tables with a predefined schema. Additionally, they support *object-oriented principals* to model relations or connection between data sets. One quite beneficial property is *inheritance* among schemes of different tables as exemplified in Figure 2.2 below. The table *Capitals* inherits the schema of the table *Cities* and extends it by additional columns. In this example the outstanding feature is that every entry made in *Capitals* will be automatically inserted in the table *Cities* with the required values for the defined schema. While the vice versa entries in *cities* are for that table exclusively since *capitals* can be considered as an extension of the schema the table *Cities* and might require more values for a valid insert operation. Since the rest of the properties is quite similar to relational DBs (see Sect. 2.2.1) we just depicted a simple example which shows the difference. The code for the described scenario is shown in Listing 2.1. In this work we will use *PostgreSQL* as a representative this type of databases.

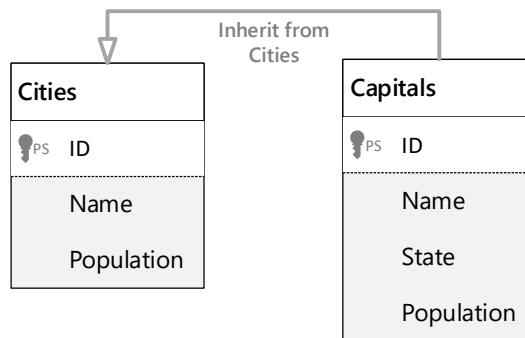


Figure 2.2.: ORDB Data Model, adapted from [Gro09]

---

```

1 CREATE TABLE cities (
2     ID      int,
3     name   text,
4     population int
5 );
6
7 CREATE TABLE capitals (
8     state  char(15)
9 ) INHERITS (cities);
  
```

---

Listing 2.1: PostgreSQL Inheritance Example, adapted from [Gro09]

### 2.2.3. Column-Family Stores

One of the most popular column-family stores is *CassandraDB*. The information we deliver in this section is based on [SF12]. Once developed by *Facebook* it became an open-source project and is currently managed by *Apache*<sup>1</sup>. The biggest data-container of Cassandra is the *keyspace*, where it puts *column families*. A column family is a collection of *rows* of similar content. Each row is identified by a *row key* and contains *columns* of which each stores a *name-value-pair*. Values can be a map of columns in which case we call the column containing this map a *super column*. A column family created of super columns is called a *super column family*, which can be considered as an equivalent to tables of RDBMS. This data structure is illustrated in Figure 2.3.

The rows belonging to a particular column family can have different columns which can be added at any time. This property makes it easier to customise the data structure while runtime. Besides the difference of the data structure compared to the relational model there are also idiosyncrasies in how the data is managed and requested. Cassandra has its own

---

<sup>1</sup><http://apache.org>

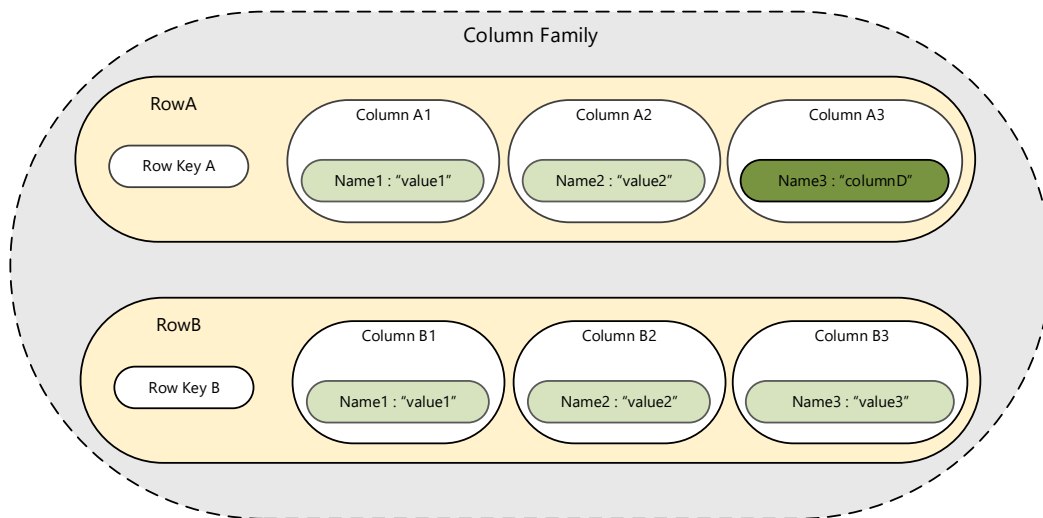


Figure 2.3.: Cassandra Data Model, adapted from [SF12]

query language, called *Cassandra Query Language (CQL)*, which has syntactical similarities with SQL. Some operations, such as joins, known from SQL are not supported by CQL. Furthermore, Cassandra supports different replication factors for the data that is written as well as a minimum number of nodes to respond on a read request, which can be set. Anyway there is always a tradeoff between consistency and availability regarding the *consistency level*<sup>2</sup> you set. According to [SF12] a reasonable compromise is the level *Quorum*, up if writes or reads are more important for your needs.

#### 2.2.4. Document Databases

Document databases store data sets in documents of types like XML, JSON, or BSON and so on [SF12]. In this section we will stick on the information given by [SF12]. One famous representative is *MongoDB* stores its document in BSON format. Each document is specified by the field *\_id*, which has to be a *Universally Unique Identifier (UUID)*, like an *ObjectID*. MongoDB's query language is expressed via JSON and has some constructs which have an equivalent in SQL. A document can be considered as a row in an RDBMS. The data model of document databases allows to nest sub-documents within documents, that again increases the performance and the ease of access. Additionally documents can have different schema, but still belong to the same collection. A collection can be considered as equal to a table in an RDBMS. Furthermore, documents can be defined at any time and do not have to follow a certain schema, if some fields are contained in one document but not in the other, then there

<sup>2</sup>[http://www.datastax.com/documentation/cassandra/2.1/cassandra/dml/dml\\_config\\_consistency\\_c.html](http://www.datastax.com/documentation/cassandra/2.1/cassandra/dml/dml_config_consistency_c.html)

is no empty field, hence a *null-value* cannot be returned. A view on the data model is depicted in Figure 2.4.

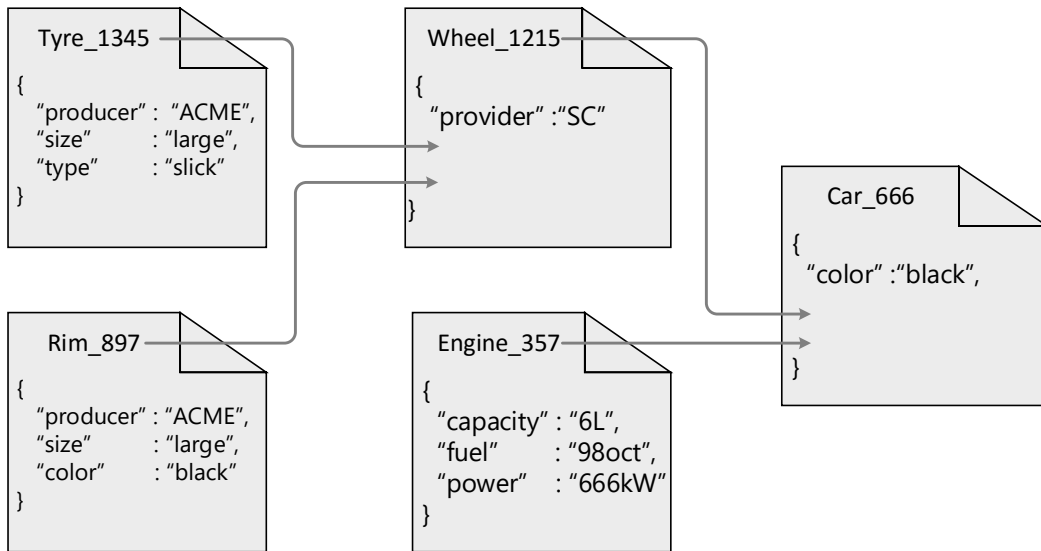


Figure 2.4.: MongoDB Data Model

This example is quite similar to the one in Section 2.2.1. Since each document has a UUID it can be considered to have the same role like a PK within an RDBMS.

So it illustrates the "Tyre\_1345" and Rim 897 as a part of "Wheel\_1215". Which in turn is, as well as "Engine\_357", is a part of "Car\_666". It is important to mention that a nested document with a certain UUID has not, unlike in RDBMS, to follow any structural or substantial restrictions besides the one given by the MongoDB engine.

### 2.2.5. Key-Value Stores

This type of stores are the simplest NoSQL stores we introduce according to [SF12]. We gained our knowledge of the topic in this section from [SF12]. For examples we will use *Riak*<sup>3</sup>, since it is open source and has been developed since 2009.

Basically key-value stores have three different types of data fragments. The largest one are *buckets*, which can be seen as name spaces for *keys*, which are used to identify content value. The actual content is stored as *value* aside the key identifying it. To access and manipulate the data there are just three basic operation, *PUT* to write the value of a key, *GET* to read value of a key, and *DELTE KEY* to delete a certain key from the store.

---

<sup>3</sup><http://basho.com/products/#riak>

On the one hand key-value stores support data structuring in various formats, such as blob, text, JSON, XML and so on. In Riak the *Content-Type* can be specified in a *POST* request. Through this simple design requests are performed highly efficient. On the other hand the stored data has to be simply structured as well, since data relations cannot really be modelled in such stores. Another downside are the queries which have to be simple as well. For example the most key-value stores only support key searching by default wherefore it is difficult to find values without knowing their keys and demands to fetch the data from the store and search for it on the local machine. Luckily Riak supports a more sophisticated mechanism to inspect data which allows to query data like you would use *Lucene*<sup>4</sup> indexes. Figure 2.5 illustrate a simple key-value store structure.

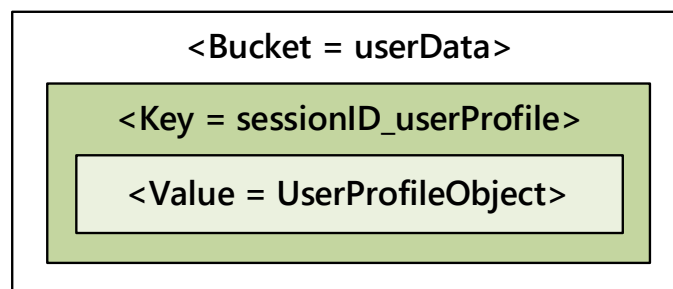


Figure 2.5.: Key-Value Store Data Model, extracted from [SF12]

In this example the bucket "*userData*" contains the key "*sessionID\_userProfile*" which identifies the value "*UserProfileObject*". Another feature of this structure is that the context of the contained data is prefixed to the key which simplifies value search.

A further important property of Key-value stores, such as Riak, allow to control the aspects of the *Consistency, Availability, Partition tolerance* (CAP) theorem. For instance Riak has two major approaches to resolve update conflicts. Either the newest write wins, or all values are returned to the client which allows resolution by implemented criteria or even manually. We won't use key-value stores in this work. Hence we will drop further characteristics and a more detailed introduction of such stores. More detail with respect to key-value stores can be found in [SF12].

## 2.2.6. Graph Databases

This section handles *graph databases*. We give an overview of the most significant properties of such stores which we elaborated from [SF12]. A graph database has two elements to store entities and the relations between them. Entities can be considered as *nodes* having desired properties and can be compared to objects in applications [SF12]. But the even more important elements are the *edges* which represent relationships between the entities but can have properties themselves as well. These properties of relationships can be used to query

<sup>4</sup><http://lucene.apache.org>

the graph but that leads to a high effort to model the relationships in the domain. One should always keep in mind is the high significance of the relations' directions [SF12]. If we take a look at the example depicted in Figure 2.6 the entity *Jeremy* has the relationship *repairs* to the entity *Car*, but vice versa it would be absurd, especially if we consider this relationship as a real life situation. There are no restrictions for the amount or the type of relations between entities. Everything that is relevant for the model can and will be modelled in the graph. For instance there are multiple relations between the entities *Richard* and *James*, where again the direction of the relations must be considered carefully.

Queries in a graph database are actually traversing of the graph and since the relationships are persistently stored they do not have to be calculated during query time which makes data inspection and manipulations very fast.

Another great benefit of graph DBs compared to RDBMS (see Sect. 2.2.1) is that they do not require a predefined model, such as a DB schema, and can be changed dynamically. For instance if we want to add an extra relation we are free to do so, without having to change the whole DB model like RDBMS would claim it. The absence of schema gives us the ability to have various context within one graph without having to define a certain global context, in the form of a schema, each time we extend our application or contextual information stored in the DB. The only thing note is the obligation of a *start* and *targen node* of a relation since no "dead end edges" are allowed in a graph.

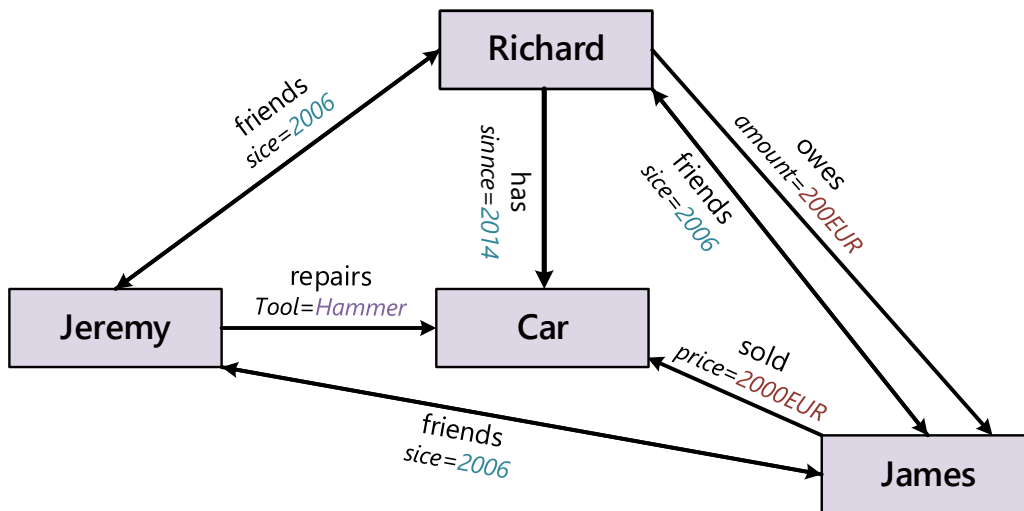


Figure 2.6.: Graph Data Model

Nowadays there are several graph DBs available, we will broach *Neo4J*<sup>5</sup> since it is widely distributed. Certain characteristics will be introduced to give a slight insight of the functionality. It allows indexing of nearly everything that has a value in a graph for the sake of faster and more comfortable opportunity to search for data inside a graph. The used service for this approach is *Lucene*. An example for such an indexed query is given in Listing A.2 which

<sup>5</sup><http://neo4j.com>

references the model from Figure 2.6.

---

```

1 Node richard =nodeIndex.get("name", "Richard").getSingle();
2 Node james = nodeIndex.get("name", "James").getSingle();
3 PathFinder<Path> finder = GraphAlgoFactory.shortestPath(
4     Traversal.expanderForTypes(OWES,Direction.OUTGOING)
5     ,MAX_DEPTH, 15);
6 Iterable<Path> paths = finder.findAllPaths(richard, james);

```

---

**Listing 2.2:** Neo4J Indexed Search Example, Changed From [SF12]

The first two lines in the Listing search the nodes by their indexed property "name". In the first line we search for the value "Richard" and in the second corresponding for "James". In this example line three to five is significant, where we specify the incoming parameters for the method `expanderForTypes`. The first argument is "OWES" and specifies the relation we are searching for. The second value "Direction.OUTGOING" defines the direction of the relation, other possible values are "INCOMINT" or "BOTH". The method is specified as follows: `shortestPath(Path Expander exp, int maxDepth, in maxHitCount)`<sup>6</sup>

It returns an algorithm which finds the shortest path between two nodes. In line six we search with the returned algorithm for all paths between the nodes `richard` and `james` which are of the type "OWES". Although the nodes have two edges between them the relationship "FRIENDS" won't be returned since it does not fulfil the specified criteria.

Neo4J also supports the query language *Cypher*. Nevertheless, there are quite many benefits of graph DBs, but there is a pretty serious downside. Graph DBs are not aggregate-oriented, but relationship-oriented and any given node can be connected to any other, hence sharding is difficult. To scale such a DB a common way is to load the working set of nodes and relationships entirely in the memory. But still the set to work with has to fit in a realistic amount of *Random Access Memory* (RAM) [SF12].

## 2.3. Data Migration

Migration is not just a simple task which can be done rashly within one step, like a "copy and paste operation". It is rather a process that requires careful consideration and strategies to avoid disasters by moving an entire application or even just a layer of it. Within this section we will describe the main challenges of data migration, as well as their causes in Section 2.3.1. We will not discuss the economic influences and impacts of data migration, since it is not a part of our work. For the one who desire more detailed information on practical data migration in general we recommend [Mor12] for further reading. Later on in Section 2.3.2 we will introduce a "*Cloud Data Migration Support Tool*" including its methodology and decision support initially developed by Bachmann [Bac12]. We will refer to this tool as *CDMT* for the sake of comprehension.

---

<sup>6</sup><http://neo4j.com/docs/stable/javadocs/org/neo4j/graphalgo/GraphAlgoFactory.html>



### 2.3.1. Challenge

As already indicated in Section 2.2 there are several types of *Database Management System(s)* (DBMS) with quite different data models. In Table 2.1 are the four most important for this work outlined. The greatest distinctions between them are recognisable in the columns "Type" and "Data Structure Definition".

Name	Storage Type	Data Structure Definition	Query Language
MySQL	rational	DB schema	SQL
PostgreSQL	object relational	DB schema	SQL
Cassandra	column family	dynamically while runtime	CQL
MongoDB	document	dynamically while runtime	JSON expressions

**Table 2.1.:** Overview of DB Types

While *MySQL* and *PostgreSQL* require a DB-schema, which defines restrictions of entries and the relation between them, *Cassandra* and *MongoDB* do not have mandatory requirement of a predefined data structuring.

A more detailed consideration reveals a possible problem even between the migration from *MySQL* to *PostgreSQL* and vice versa. Although both use query languages based on the SQL standard and require a DB schema *PostgreSQL* supports inheritance within the schema and as relation between tables. This aspects for instance often claims refactoring steps in the DB-schema and the *Data Access Layer* (DAL) to provide a proper data handling without changing the concept of the representation layer.

Besides the obvious differences of querying the considered DBs there is a even bigger obstacle, the different ways how data is stored in the respective DB, more details are in Section 2.2.

Since each of this DBs has an own data model there is no simple, generally valid way to migrate data from one system to another. Hence, it is recommendable to use an already developed methodology which covers the indicated scenarios with a solution support for upcoming problems and challenges. There are nowadays some products on the market with the required functionality, like *AWS Database Migration Service*<sup>7</sup>, but it is only limited to AWS as target platform.

In the following Section 2.3.2 an introduction and a short overview on an general methodology is given.

### 2.3.2. Migration Tool and Methodology - Bachmann

A major problem which led to this work was that there were several services to migrate data to the Cloud, but they were just for several specific Cloud service solutions mostly.

Moreover there was not any support for migration between the respective Cloud services. A further difficulty was that there was not a methodology to classify a migration scenario in context of Cloud migration. The goal was to develop a holistic method to migrate the

---

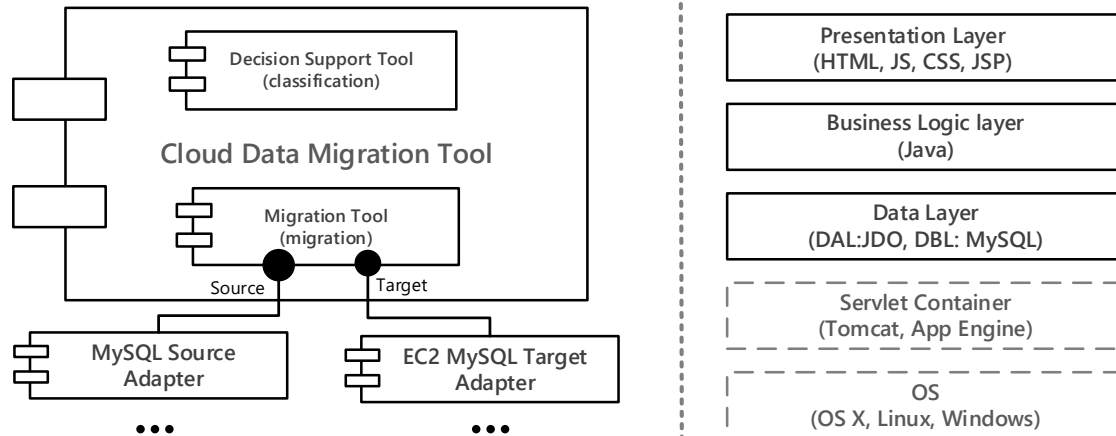
<sup>7</sup><https://aws.amazon.com/dms/>

*Database Layer (DAL)* into the Cloud.

As part of his diploma thesis Bachmann developed, besides the methodology, a Java Web application for data migration. It includes a decision support and some drivers to migrate data properly.

### Application Structure

CDMT consists of two main components, as illustrated in Figure 2.8. The component *Decision Support Tool* helps the user to identify the most fitting solution for his needs. The second component *Migration Tool* migrates the data after the user has configured his desires and in some circumstances adapted his DAL and his *Business Logic Layer (BLL)* to the new requirements for data usage. It contains different adapters for respective data stores and hosting solutions. In order to be extensible more drivers can be added if necessary. The architecture of the application is layer based and is composed of three layers. The *User Interface (UI)* is included in the *Presentation Layer* and combines several technologies, but is mostly realized with *Java Server Pages (JSP)*s (see Fig. 2.7). Beneath is the BLL which is entirely implemented in Java. The lowest layer is the *Data Layer* that can be decomposed in further components. The layers which do not belong to the application, but are necessary to run it are represented with dashed borders.



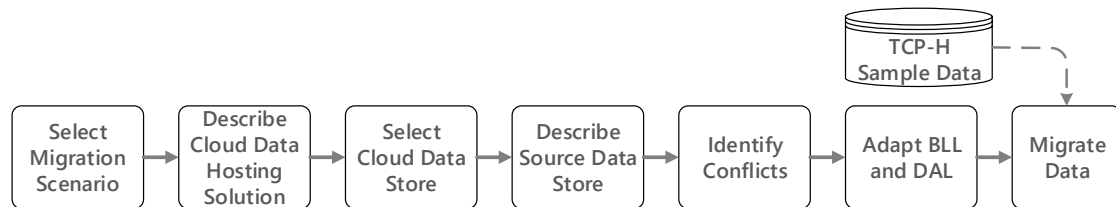
**Figure 2.7.:** System Component Diagram and Layer Diagram of CDMT, adapted from [Bac12]

### Migration Methodology

The methodology defines a process which consists of eight major phases as depicted in Figure 2.8. The first seven phases demand a high degree of user interaction, since they are the main part of the classification process. They can be fragmented in sub steps. The first four phases (*Select Migration Scenario*, *Describe Cloud Data Hosting Solution*, *Select Cloud Data Store*,

## 2.3. Data Migration

*Describe Source Data Store*) are based on questionnaires to limit possible options for users needs.



**Figure 2.8.:** Methodology for Data Migration, adapted from [Lam13]

To give an example for a better comprehension the defined *Data Migration Scenarios* are summarized in Table 2.2. They form the alternatives of the first phase and were an essential part of that work. The last phase (*Migrate Data*) is automated and executed by the tool.

Scenario	Description
<b>Plain Outsourcing</b>	Migration of local DBL to the Cloud without changing the datatore type. Whole application is migrated, data migration is a part of overall Cloud migration.
<b>Usage of Highly Available Data Store</b>	Migration from a non-highly-scalable to a highly-scalable. Cloud data store. Usually from From a RDBMS to a NoSQL or a Blob store.
<b>Geographic Replication</b>	Data is moved close to the processing unit in order to reduce latency. Whole Application is migrated to the Cloud as well.
<b>Data Distribution(Sharding)</b>	Data is split to disjunct partsand moved to different data centers. This might reduce latency by moving data closer to the client and additionally increase horizontal scalability.
<b>Off-Loading of Peak Loads (Cloud Burst)</b>	Moving data temporarily to the Cloud to deal with hight traffic situations. Usually the whole application is moved. It is moved back when the traffic decreases and can be handled.
<b>Work on Copy of Data (Data Analysis)</b>	Cloning the data to perform recourse intense analysis without having influences on the production server.
<b>Data Synchronization</b>	Temporarily access to the DBL to duplicate content locally. Changes might be done off-line and be synchronized as soon as they are ready, or de network is available again.
<b>Backup</b>	Creates plain copy of the DBL in a certain state and stores it to the Cloud. Usually used to recover failed systems.
<b>Archive</b>	Similar to Backup. The entire or parts of the DBL can be copied and stored in the Cloud at a certain point.

<b>Data Import from the Cloud</b>	Creating a local copy from a Cloud service by accessing it via API.
<b>Data Usage from the Cloud</b>	Outsourcing of the DBL to the Cloud only. Local clients can access the data from the store remotely.

Table 2.2.: Data Migration Scenarios by Bachmann, Adapted From [Bac12]

## 2.4. Software Integration Strategies

We extracted the information in this section from [Lig09]. The containment is mostly restricted to the issues that are relevant for this thesis. Basically we consider two strategies *Bottom Up*, in Section 2.4.1, and *Top Down*, in Section 2.4.2 to integrate modules into a software system. Section 2.4.3 gives an outlook of possible combinations and of our use of these strategies. Needless to say that it is possible to use hybrid forms of these two strategies which emphasis could tend more to one of the strategies depending on the requirements that have to be met. The basic difference between these two integration techniques is the order in which the functionality of developed parts to integrate is validated and evaluated. Generally missing invoking modules are substituted by drivers and the invoked ones are replaced by dummies mostly with default return values. The consequence of such an approach is increased independence during the period of development and possibly a reduction of time needed for progress since many tasks can be handled concurrently. The following subsections illustrate the corresponding procedures in a simplified manner.

### 2.4.1. Bottom-up Integration

The purpose of bottom-up integration is to set up a system from the atomic parts providing basic functionality of the system. In this attempt the access to the individual modules and their responses to it are tested. It could be a simple function call which return values are evaluated by the test driver of a more sophisticated testing environment as well.

In the description below we consider test scenarios with usage of drivers. The main precondition for starting the entire integration process, including the testing, is to assure that all drivers are working as anticipated.

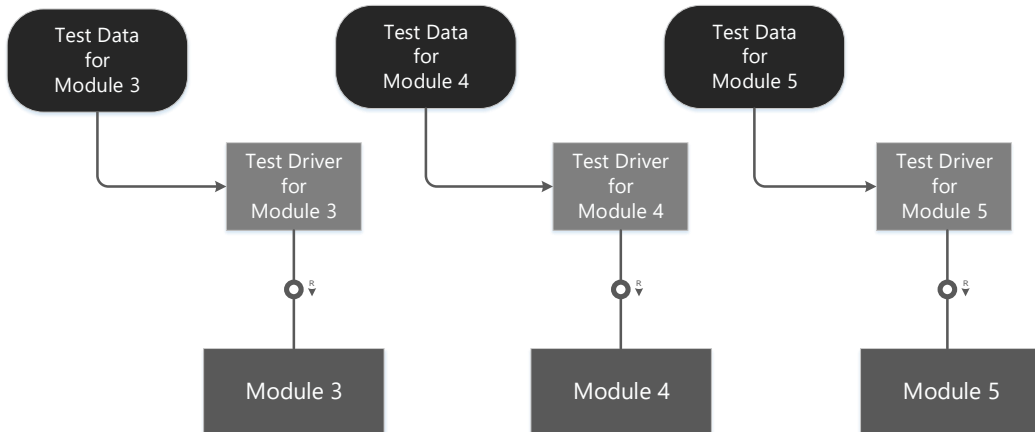


Figure 2.9.: Isolated Modules Bottom-up Test, Adapted From [Lig09]

As soon as a module passes all indispensable tests its implementation is assumed as corresponding to the specification and its returned results as correct. Once the loose modules passed the test they are joint to components. The structure of the components aligns on the system architecture. As illustrated in Figure 2.10, the behaviour if the whole component is investigated by accessing it through its interface, in this case, represented by *Module 1* and *Module 2*.

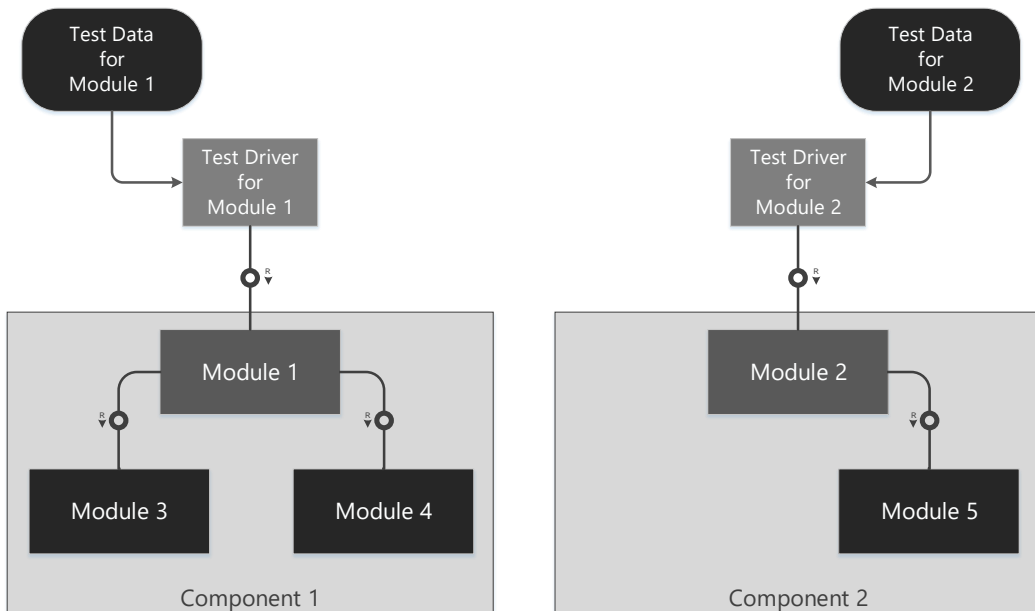


Figure 2.10.: Isolated Components Bottom-up Test, Adapted From [Lig09]

Such interfaces usually receive request by relying on the assumption from the previous step. The modules for rudimentary functionality can be excluded as cause for failures possibly occur

during this stage. Apart from that there is no need to search for corrupt data transformation inside the call hierarchy above the interface, because the test data is handed over from the test driver to the interface immediately. Therefore the only source of misbehaviour remains within the interface itself. Considering this circumstance saves a lot of time to detect and locate bugs.

In this final state of integration the properties of every part included in the system are reviewed and evaluated as suitable in respect to their performance. Finally the inspected components can be integrated into a complete system. The only outstanding point is the evaluation of the whole system itself, as drawn in Figure 2.11. The only elements of the systems, which are not proved, are interfaces for access from the outside. At this point any suitable testing process can be used.

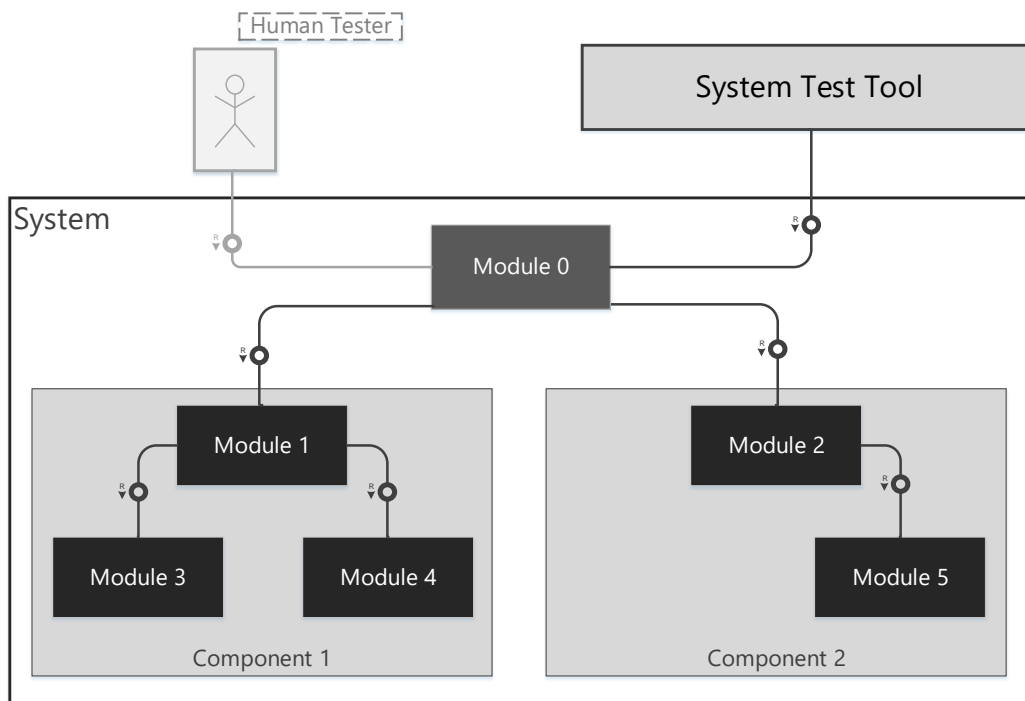


Figure 2.11.: Overall System Test, , Adapted From [Lig09]

### 2.4.2. Top Down Integration

The main difference to bottom-up integration 2.4.1, indicated in the previous subsection, is the sequence of testing and the intention of what is to prove. Furthermore, the Top Down Integration not necessarily uses drivers to pass test data to invoked modules.

Instead it is essential to provide dummies simulating the services of modules which appear at a lower position in the call hierarchy. As pictured in Figure 2.12, the first module to test is the topmost interface of the System, for example a *Graphical User Interface* (GUI).

The intention of this step is to analyse if request put to the system are delegated to the right module. In addition, it is tested if the module is accessing services and functions, provided by other parts of the system, as expected.

When the module building the root of the call tree successfully completed all tests the dummies are exchanged by the real modules invoked, which are tested in a similar way as the modules above, as shown in Figure 2.13.

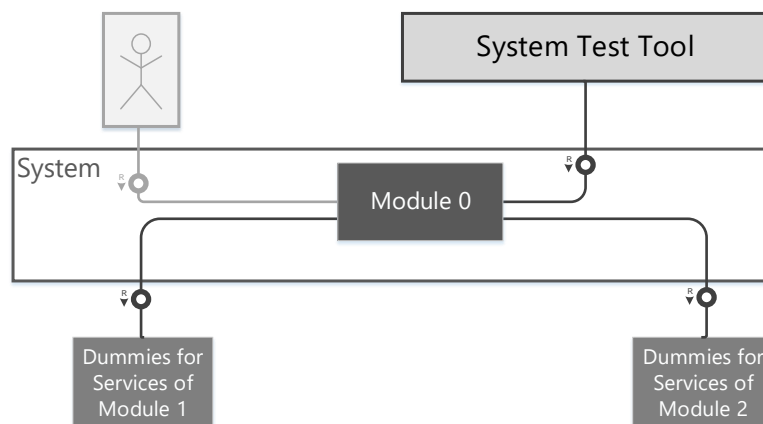
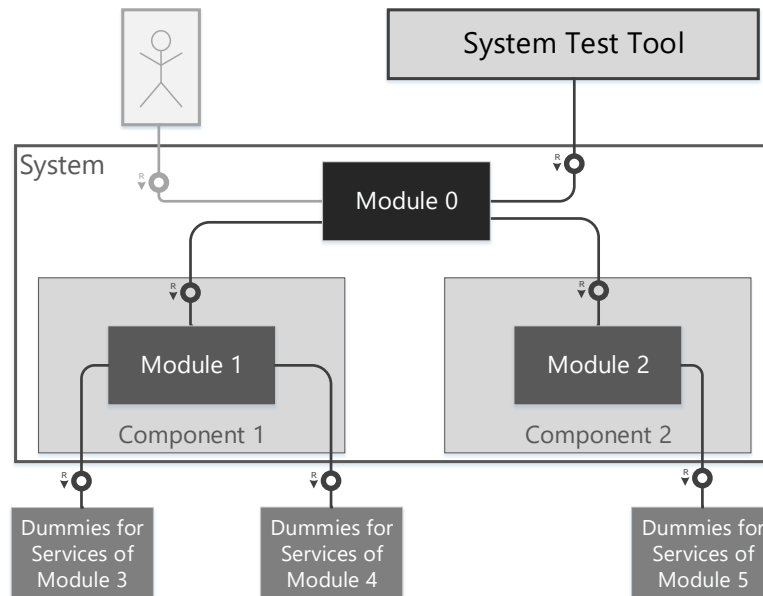


Figure 2.12.: Testing Highest Invoking Module, Adapted From [Lig09]

We consider the integrated modules in this step as interfaces of components to stay in the metaphor from the previous subsection and handle in this description as much scenarios as possible. Since the correctness of the invoking module and the required behaviour of all dummies is verified failure location can be restricted to the test candidates. Eventually in the last step of this integration approach the modules providing rudimentary functionality and services are integrated into the overall system and the work specific for top down integration is completed.



**Figure 2.13.:** Testing Components' Highest Invoking Module, Adapted From [Lig09]

As well as, the structure is alike the one shown in Figure 2.11 in the earlier subsection, the open tasks are similar. Before deploying the entire system must be tested to assure that all integrated parts are working correctly in synergy with each other. The fundamental distinction to bottom-up integrated systems is that the parts invoked last in the call hierarchy are inspected implicitly within the system test. As these are the only element of whole system, whose quality is not evaluated, they are also regarded the only possible malfunction source in this state.

### 2.4.3. Conclusion

The strategies introduced previously in Sections 2.4.1 and 2.4.2 come in handy for software within the development process. Having an already deployed or running system can benefit from both attempts if an additional module is needed to be integrated for whatever reason. The newly developed module can be tested isolated by using test drivers with test data as input to detect malfunctions before they can have an effect on the whole system.

Once the module is classified as stable and usable it can be integrated in the overall system. As the final step the entire system with the newly integrated module should be tested against the specified requirements.

Such an approach is appropriate for critical systems in respect to downtime. Furthermore it delivers the ability to prove the inner qualities of the module without being influenced by possible side effects of its runtime environment. That again eases error searching and causes of it. In our work we will use an existing system which we extend by components from different concurrent versions. Hence, we will use both strategies for this purpose in



dependency to which layer an integrated component belongs. Besides we will develop some components in this work as well. We will integrate them by using the top-down approach.

## 2.5. Java Extension Mechanism

Many software products provide the possibility to extend their functionality without changing their previous source code. The term to offer such functionality is add-on. These software components can be loaded while runtime to the core system which functionality should be extend. In this section we introduce the principals of a mechanism one can use to enable extensibility by components in order to extend its functionality.

Since the business logic of the Web application CDMT is implemented in Java we will restrict the content of this section to the most common way to offer extensibility via add-ons in Java. We gained our knowledge of this topic from Java's documentation [Cor] which is provided by Oracle<sup>8</sup>. For a better comprehension of this technique you must know how the *Java Runtime Engine* (JRE) loads classes in general, which we introduce in Section 2.5.1. The next step is to leverage this mechanism to load classes, or components, while runtime, which we explain in Section 2.5.2.

### 2.5.1. Class Loading

The basic idea of Java platform is that every class loader has a "parent" class loader except of the Bootstrap class loader, as depicted in Figure 2.14.

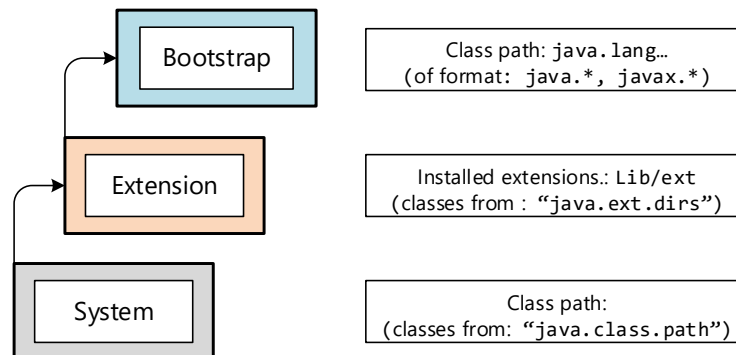


Figure 2.14.: JRE Class Loader Hierarchy, Adapted From [Yim]

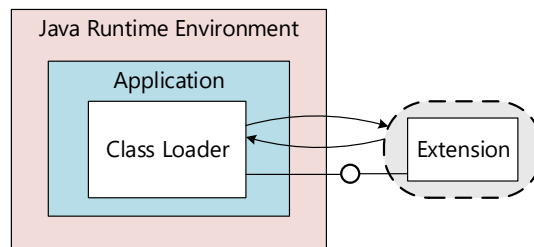
The Bootstrap class loader is responsible for loading the runtime classes. Afterwards the Extension class loader is in charge to get classes in JAR files from the lib/ext directory of the JRE. The last one is the System class loader which is responsible for the classes on paths specified by the system property `java.class.path`.

<sup>8</sup><http://www.oracle.com/index.html>

Furthermore "the Java platform uses a delegation model for loading class"<sup>9</sup>. When a class loader has to load a new class it first delegates the request to its parent loader to find out if this class is already loaded. In case the parent does not return the required class it attempts to find the required class itself. If the class could not be found by any class loader a `ClassNotFoundException` is returned.

### 2.5.2. Extension Loading

It is always beneficial to keep a software product customisable for the user. One common way to provide this possibility to load extensions, further referenced as add-ons or plug-ins, after deploying the software. In this section we present the principal how to enable such extensibility for Java applications. One major requirement is to be able to load add-ons dynamically as Figure 2.15 depicts.



**Figure 2.15.:** Schematic Functionality of a Class Loader

As we explained there are basically three types of class loader (see Sect. 2.5.1). But the disadvantage is that components to be load have to be declared in advance, to be used in an application. To be able developing add-ons we first have to define a service which interfaces are later implemented by the extensions. For the sake of simplicity, we assume just to need one kind of add-on, which only claims to implement only one interface. Listing 2.3 shows a suggestion of an interface definition, which is needed to be able to use the functionality of the add-ons once they are provided. Listing 2.4 outlines the skeleton of an add-on implementation that uses the provided interface.

---

```

1 public interface PlugIn {
2     public String getPluginID();
3 }

```

---

**Listing 2.3:** Fragment of Interface Definition

<sup>9</sup><http://docs.oracle.com/javase/tutorial/ext/basics/load.html>

## 2.5. Java Extension Mechanism

---

```
1 public class connectorPlugIn implements PlugIn{
2     @Override
3     public String getPluginID() {...}
4 }
```

---

**Listing 2.4:** Fragment of Add-on Implementation

Finally, as the application is prepared to use add-ons and some of them are deployed we can access it by using one of Java's provided class loaders. We give a short example by using `URLClassLoader`<sup>10</sup>, to show the entire scope of work when attempting to load components dynamically. As Listing 2.5 we first have to extend the systems `URLClassLoader` for the sake of security, since that way it is not accessed directly. In this example we skipped a detailed explanation of how to implement a class loader, since every one can decide which security mechanism and additionally methods should be included.

```
1 public class PluginLoader extends URLClassLoader{...}
```

---

**Listing 2.5:** Extend Class Loader Snippet

As we have provided our implementation of a class loader we can now load add-ons dynamically as illustrated in Listing 2.6. We have to give the directory where our archive with the add-ons is located and add it to the system's class loader. Afterwards we can load a specific plug-in as shown in Line 5 of Listing 2.6. After this step we can use the add-on's functionality by invoking the methods from the connector object.

```
1 URLClassLoader sysLoader =
2     (URLClassLoader)ClassLoader.getSystemClassLoader();
3 PluginLoader loader = new MyClassLoader(sysLoader.getURLs());
4 loader.addURL(new URL("file:<pathToThePlugins>/connectorPlugIns.jar"))
5 Class <Plugin> connector = loader.load("pluginPackage.pluginName");
```

---

**Listing 2.6:** Snippet to Load a Plug-in

Java provides several class loaders in its basic libraries. Hence, it is recommended to do research which is the most appropriate class loader to use for a certain goal.

---

<sup>10</sup><http://docs.oracle.com/javase/8/docs/api/java/net/URLClassLoader.html>



---

## 3. Related Work

---

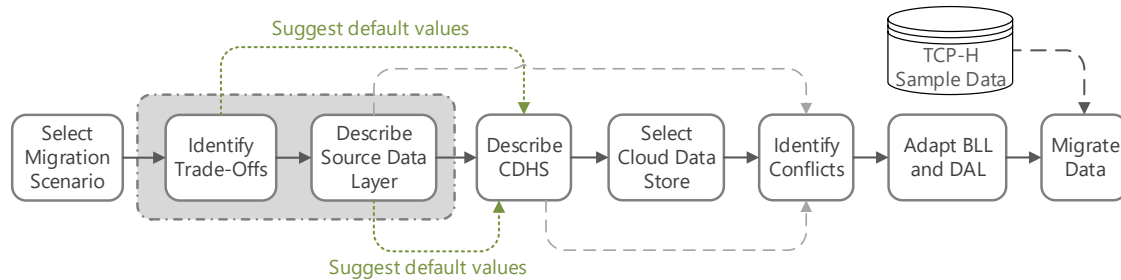
This chapter gives an overview of related work. We divide this chapter into two major subjects. As it is important and time reducing to know about the weaknesses of the system and is useful to know about already identified improvement suggestions, we we introduce already finished work on CDMT in Section 3.1. In Section 3.2, we introduce ESB<sup>MT</sup>, a research result, which an multi-tenant aware Enterprise Service Bus. Although, it is a quite another topic we can use some of the concepts for this work.

### 3.1. Previous Results

This section presents results issued to the topic of migration and delivering a foundation for our work. All publications depict within this section are based on the Bachmann's Methodology and Tool for Migration of the DBL to a cloud [Bac12] developed with in his diploma thesis.

#### 3.1.1. Migration Scenario: RDBMS to NoSQL

The focus of this thesis was the migration scenario of data from RDBMS to NoSQL stores. It is not possible to create a general model to handle this task, since NoSQL embraces different store types, Lamllari discusses in her thesis [Lam13] difference between the mentioned types. Tables of "*Application Use Cases*" for several NoSQL DBMS in that work give an overview of reasonable possibilities to use the respective DBMS, and not suitable use cases to minimize wrong decision towards the usage of a specific DBMS. Within the scope of this work an extension of the CDHS taxonomy was provided to widen the use of the whole CDMT. There are now two different migration strategies depending on if it is desired to move Data from a RDBMS to a NoSQL store or not. The scenario in the focus of this work has a varying migration strategy to the others, as depicted in Figure 3.1. The methodology for migration is based on the original one, which we introduced in Section 2.3.2 but has been customized. For example the phase *Describe Source Data Layer* is moved chronologically before the Describe CDHS, which was originally in Bachmann's definition *Step 4a*. It was added another phase *Identify Trade-Offs* which is incorporated into the second phase, but is illustrated as separate within a work group for a better recognition. Furthermore it is suggested to move the needed adaption, for example to add a a new NoSQL store to the list of Cloud Data stores, at the DAL and the BLL, when a mapping of relational to NoSQL DB-schemas is considered. Additionally locking of values that do not apply to the migration scenario are preventing conflicts by later users' choice changes. By such restrictions the CDMT migration process is cleaner and offers a clearer presentation of the UI. Besides those changes an implicit selection of possible source data stores is suggested to reduce unnecessary considerations.



**Figure 3.1.:** Extended Methodology for Scenario RDBMS to NoSQL, adapted from [Lam13]

But otherwise some for NoSQL stores not applicable categories in the taxonomy were removed. Further extensions within the CDMT are three new DB-adapters. One source adapter for PostgreSQL and two target adapters for Cassandra and MongoDB. The main changes compared to the initial version of the CDMT were in the part responsible for the "Decision Support" of the migration with the main goal to facilitate the scenario with a RDBMS as source and a NoSQL system as target.

Even if in the focus was just one migration scenario this work delivered a broad understanding of the NoSQL stores. Even more important it gave a practical point of view on the subject of data migration especially with respect to the scenario for migrating data from RDBMS to a NoSQL store.

### 3.1.2. Evaluation of Methodology for DBL Migration: Industry

An evaluation of Bachmann's Tool [Bac12] based on a case study from the industry was done by Nachev in his thesis [Nac13]. The migration scenarios of this work used a tool developed by "NovaTec Holdin GmbH"<sup>1</sup> and consists basically of three components. One of them is a web application and the other two are DBs. We cannot deliver a more detailed description since the permission for that was not gained.

The process model for the evaluation was used ITIL CSI [MMB09]. The evaluation consists of two iterations which use different scenarios and can be subdivided into further more fine grained steps. Within his work he identified three basic scenarios being relevant for his attempt:

1. Migration of both DBs
2. Migration of the Web app
3. Migration of the entire Application

Since the focus of Nachev's thesis was Bachmann's methodology, which just covers data migration, he omitted the second scenario.

<sup>1</sup><http://www.novatec-gmbh.de/en>

### 3.1. Previous Results

The steps of the first iteration are quite similar to each other (see Table 3.1). In this iteration Bachmann’s tools and methodology was used to migrate the local PostgreSQL stores to a “Virtual Machine” (VM) hosted on the on-premise Cloud of NovaTec.

	Step 1	Step 2
Migrated Component	DB 1	DB 2
Methodology	Bachmann	Bachmann
Used Tools	Bachmann’s tools	Bachmann’s tools
Source DBS	PostgreSQL	PostgreSQL
Target DBS	PostgreSQL	PostgreSQL
Source Environment	Local System	Local System
Target Environment	NovaTec Cloud	NovaTec Cloud

**Table 3.1.:** First Iteration of Evaluation by Nachev, Extrated From [Nac13]

The second iteration of this work is segmented into two steps as well. Whereas the first step compromises the whole first iteration with a different target environment.

Table 3.2 shows the structure of the second iteration with its key aspects. We will pass on describing the first step as it is analogously to the whole first iteration.

The target platform of the second iteration is Amazon<sup>2</sup> Cloud. Since in the second step of this iteration the entire application is migrated the methodology of AWS and Bachmann are combined as depicted in Figure 3.2.

An other point to mention is that Nachev did not perform the last two steps, “Leverage the Cloud” and “Optimization”, because it was not within the scope of his work.

After the both iteration he compared durations of test runs of the hosted solutions with the one measured with the ones when the NovaTec application is executed entirely local.

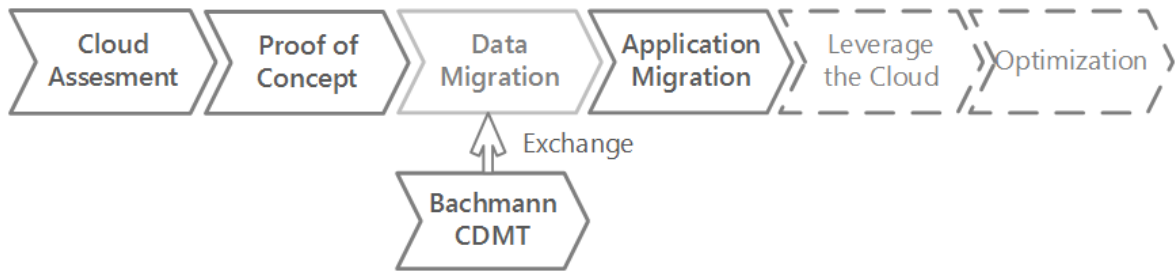
	Step 1	Step 2
Migrated Component	DB 1 & DB 2	Entire Application
Methodology	Bachmann	AWS, Bachmann
Used Tools	Bachmann	AWS, Bachmann
Source DBS	PostgreSQL	PostgreSQL
Target DBS	PostgreSQL	PostgreSQL
Source Environment	Local System	Local System
Target Environment	Amazon Cloud	Amazon Cloud

**Table 3.2.:** Second Iteration of Evaluation by Nachev, Extracted from [Nac13]

While performing the evaluation steps several problems and errors occurred. Each error was documented in detail including a classification of severity and priority. Additionally an description, suggestion for error handling and solution were given to improve Bachmann’s tool and methodology.

Besides of the evaluation Nachev extended the CDMT by implementing a source adapter and a target adapter for PostgreSQL to be able to execute his specified scenarios. Furthermore he already resolved some of the documented errors permanently by refactoring or other

<sup>2</sup><http://aws.amazon.com>



**Figure 3.2.:** Methodology of AWS and Bachmann Combined, Based on [Nac13]

programming activities.

The knowledge acquired in this thesis revealed the combination between the CDMT and the methodology of Amazon. This use case is facilitated different tools and methodology to enable the opportunity to achieve an aim which exceeds the functionality of the CDMT. Even handier is that there was no need to perform a huge amount of work within the CDMT.

### 3.1.3. Evaluation of Methodology for DBL Migration: eScience

This subsection describes briefly the work of Guo [Guo13] in which he evaluated the CDMT by migrating a scientific work flow simulator. The scope of the examined scenario is the migration to an instance of Amazon Cloud. Nevertheless the focus of this work is the migration to the comparison of Bachmann's attempt and tool to the one of *Amazon Web Service*<sup>3</sup> (AWS). It is also mentioned that two services of AWS, *Amazon Rational Database Service*<sup>4</sup> (RDS) and *Amazon Elastic Compute Cloud*<sup>5</sup> (EC2) should be used. To get more precisely the evaluation consists of two iterations. In each iteration the BLL and the DLL are migrated in respect to the used methodology. Since Bachman's methodology and tools just focus on migration of the DBL, the BLL has to be migrated manually. A brief summary of the iterations is shown in Table 3.3.

	Iteration 1	Iteration 2
Methodology	Bachmann	Amazon
Used Tools	Bachmann's tools, rsync	Bachmann's tools, Amazon tools, rsync
BLL Migration	manually	manually
BLL Target Platform	Amazon EC2	Amazon EC2
DBL Migration	Bachmann's Tools	Bachmann's Tools
DBL Target Platform	Amazon RDS	Amazon RDS

**Table 3.3.:** Envaluation Iterations by Guo, Extracted From [Guo13]

<sup>3</sup><https://aws.amazon.com>

<sup>4</sup><https://aws.amazon.com/rds>

<sup>5</sup><https://aws.amazon.com/ec2>



### 3.2. Multi-tenant Open-Source Enterprise Service Bus

---

Each iteration of the evaluation process is mapped to the ITIL CSI [MMB09] process. Basically his thesis delivers a quite detailed description for the usage of Bachmann's tool and methodology. Furthermore several errors emerged during the execution of both iterations. Each error is described in a sufficient way including strategies to handle a particular error and solutions to correct or to avoid it. This facts lead to a better comprehension of Bachmann's attempt in a more practical way. No persistence changes to the CDMT were made, since many of the occurred as part of the discussed scenario in conjunction with the used configuration and environments.

#### 3.1.4. Positioning and Distinguishes

The introduced results do not match our needs, since the focus of this work is integration. Hence our attempt has not much in common with the already published work. Nevertheless this work is based on Bachmann's CDMT and methodology, so we can benefit from the gained knowledge and results of the concluded publications.

We will use the insight of the thesis introduced in Section 3.1.3 to reduce the effort in case of emerging misbehavior while the usage of the tool.

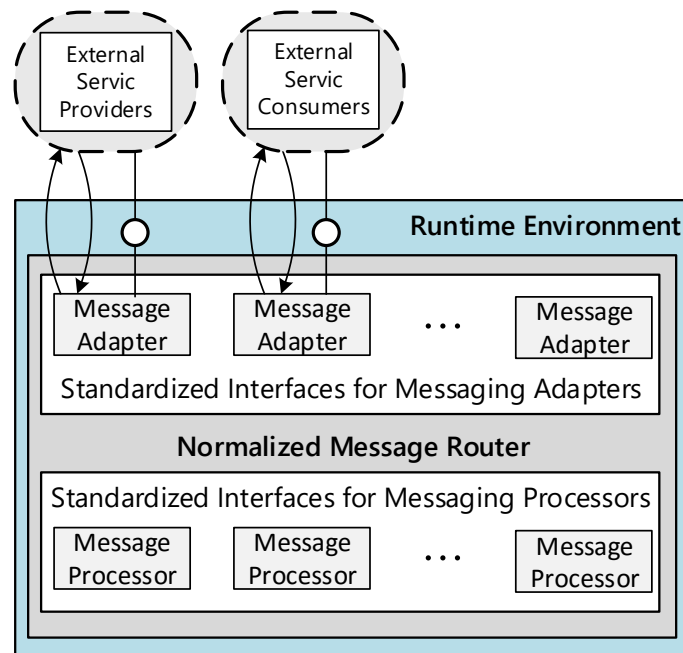
The attempt of this paper is to increase the stability and possible usage of the CDMT. On that account we will take results and implementations of the work introduced in the Sections 3.1.1 and 3.1.2 and join them to one more powerful result.

A further distinction to Nachev is, that we won't use the toll and methodology as part of an overall migration. Solely some DBL migration to different types of storages will take place as part of the validation (see Sect. 7). Furthermore we will resolve some outstanding issues to provide a more pleasant use of the CDMT.

## 3.2. Multi-tenant Open-Source Enterprise Service Bus

A core concept to integrate application is the *Enterprise Service Bus* (ESB) technology, whereby it is acting as the messaging hub between applications. It is used as a core component of each *Service-Oriented Architecture* (SOA).

In this section we take a look on ESB<sup>MT</sup> which is introduced by Strauch et. al in [SALM12]. One major task of that work was to enable multi-tenancy of the ESB benefit from Cloud computing paradigm in an business environment. A further important aspect of ESB<sup>MT</sup> is the extensibility for the needs of every user by integrating external services to without having to change the original source code. ESB<sup>MT</sup> extends the open source Apache ServiceMix solution. However, we will just take a look on aspects of this work which can be reused within this thesis. As Figure 3.3 shows every ESB instance is able to use services of external providers and offer them to external service consumers.



**Figure 3.3.:** Architecture of an ESB Instance, Based on [SAGSL13]

Since not each user needs to use every available service that is accessed by an ESB it benefits from properties of multi-tenancy. Every ESB<sup>MT</sup> instance is executed in a *Java Business Integration (JBI)* environment and belongs to a *JBI Container Instance Cluster*. The extensions are integrated by the mechanism of bindings. We will reuse the concept of extensions while runtime and the multi-tenancy concepts in a some different way since the set-up of the ESB<sup>MT</sup> is quite different from the one of the CDMT. Since the CDMT does not extend Apache ServiceMix and does not use JBI libraries we must consider an other mechanism to provide extensibility. Furthermore, the concept of multi-tenancy properties cannot be implemented in the same extend as introduced in context of ESB<sup>MT</sup>, since the the CDMT only supports different user, but does not subdivide it by tenants or similar groups.

However, we will reuse the idea that not every user should have the access to every single component the CDMT provides, rather it is important that not all dynamically loaded components should not be provided to all users.

---

## 4. Concept and Specification

---

In the following chapter, we will specify the changes of the CDMT (see Sect. 2.3.2), as well as their requirements. Since the work is ongoing we will reuse some results from previous work to avoid redundant labour. First of all, the implementation is separated in two iterations to reduce the risk of failure. In Section 4.1 we will introduce the implementations ought to be combined to benefit of the respective progress, which is the first iteration. Mainly we will use draw on the achievements of Lamllari [Lam13] (see Sect. 3.1.1) and Nachev [Nac13] (see Sect. 3.1.2).

Furthermore it will give an overview over open points, like misbehaviour of the UI. Additionally, some functions of the CDMT, which are still behaving insufficiently, will be specified. The descriptions will be as detailed as necessary and brief as possible to avoid repetitions, since we already introduced these in Section 3.1 .

Afterwards, we introduce requirements for the behaviour of the add-on mechanism, which extends the functionality of CDMT, in Section 4.2. We based define this part of the specification with Use Cases. Finally, we explain the specified non-functional requirements in Section 4.3. Since they are generally valid and will be considered during all following tasks, we formulated them in a separate section. Here after we will reference the achievements of the authors as follows:

- B:** Adaptations, implementation and suggestions done by Bachmann.
- L:** Adaptations, implementation and suggestions done by Lamllari.
- N:** Adaptations, implementation and suggestions done by Nachev.

In this way we avoid to take credit for work that was not done by ourselves. Also it enables the reader to identify the source and the originator of suggestions and the work used in this thesis.

In Section 4.2 we specify the add-on extension mechanism and requirements towards it.

### 4.1. Integration and Adaption

In this section will the tasks specified do be done as part of the integration. In Section 4.1.1 we will give an overview of the DB adaptors to unite within our prototype. Section 4.1.2 will specify functional requirements the software shall meet after the integration. For this phase we will use the prototype implemented by Bachmann and adapted by Nachev es our base-version. Each introduced section is to consider as a separate step of the overall integration process.

Since Lamllari [Lam13] and Nachev [Nac13] implemented their improvements concurrently and without any exchange now there are two different parallel versions ot the CDMT. As

both made their own and distinguishable adaptation as part of their theses the respective focus was quite different.

#### 4.1.1. Adapters

This section will introduce the adapters that are existing in the different versions of the CDMT and might strictly be considered as separate and exchangeable components.

The different provided adapters are summarised in Table 4.1 and ordered by their respective type and author.

	<b>Bachmann</b>	<b>Lamllari</b>	<b>Nachev</b>
<b>Source Adapter</b>	MySQL	PostgreSQL	PostgreSQL
<b>Target Adapter</b>	Google Cloud SQL <sup>1</sup>	Cassandra	PostgreSQL
	Azure SQL <sup>2</sup>	MongoDB	
	EC2 MySQL		
	Local MySQL		

**Table 4.1.:** Overview of Provided Adapters

It is obvious that the source adapter for *PostgreSQL* is implemented twice. For our approach we will use the one with which proves itself having a cleaner implementation in a code review. We will skip the validation of functionality before the integration, since it already was provided by fact that each of the adapters already was tested by the respective author within his work. After the integration process the source and target adapters ought to work in the implemented way. Therefore we will inspect the capability of the integrated adapters implicitly in a system test, since changes of the whole system could have side-effects on components which were not changed themselves. Respectively the source adapters shall connect to the specified store types and readout data. On the other hand the target adapter have to connect to the specified target store types and write data in the store.

#### 4.1.2. Functional Requirements

This section contains functional requirements, which were specified by the previous work, or emerged while the usage of one of the prototypes. However Table 4.2 includes the identified requirements ordered by their *ID* with a brief *Description*.

Whereas the ID type is specified as follows:

$$ID = Type - Nr(Author) \mid Type \in \{FR, NFR\} \wedge Nr \in \mathbb{N} \setminus 0 \wedge Author \in \{B, L, N\} \cup \{\}$$

<sup>1</sup><https://cloud.google.com/sql>

<sup>2</sup><https://azure.microsoft.com/en-us/services/sql-database>

#### 4.1. Integration and Adaption

---

ID	Requirement	Description
FR-1(L)	Add properties to the category <i>Select Querying Features</i>	The following properties are to implement: -SQL-like query language -Aggregate queries support -RegExp queries support -Full-text search -Secondary indexes -Geo-spacial indexes
FR-2(L)	Delete redundant option in <i>Step 1a: Select Migration Scenario</i>	The option <i>Data Usage from the Cloud</i> is a sub-scenario of <i>Plain Outsourcing</i> . It will be removed from the CDMT.
FR-3(L)	<i>Step2:Identify Trate-Offs</i>	In case the scenario " <i>RDMBS to NoSQL</i> " is selected a step to identify trade-offs of different NoSQL store types is shown.
FR-4(L)	Show always <i>Steps 1a, 1b and 1c</i>	<i>Steps 1a, 1b, 1c</i> have always be selectable independently of the selected migration scenario.
FR-5(L)	Adapt work flow of the methodology	The step <i>Describe Local Data Layer</i> has to in the work flow in any case be before the step <i>Describe Desired CDHS</i> .
FR-6(L)	Adapt category <i>Data Constraints</i> in <i>Step2: Describe Desired CDHS</i>	Property <i>Max Index Size</i> should be added.
FR-7(L)	Extend <i>Data Constraints</i> in <i>Step2: Describe Desired CDHS</i> by category <i>Caching</i>	Category <i>Caching</i> should be extended by adding the properties <i>Direct Caching Support</i> (options: Yes/No) <i>Caching Layer</i> (options: Built-in/ Support for integrating Caching Layer on top of data store).
FR-8(L)	Accomplish category <i>Management and Maintanance effort</i> in <i>Step2: Describe Desired CDHS</i>	Property <i>Communigy Support</i> should be added.
FR-9(L)	Tool tip for category <i>Data Compression</i> in <i>Step2: Describe Desired CDHS</i>	The currently empty tool tip shall show helpful information on data compression to support users choice.
FR-10(L)	Tool tip for category <i>Querying</i> in <i>Step2: Describe Desired CDHS</i>	The currently empty tool tip shall show helpful information on querying to support users choice.
FR-11(L)	Changes in category <i>Consistency, Availability, Partition tolerance (CAP)</i> in <i>Step2: Describe Desired CDHS</i>	Renaming of property <i>Availability in case of Partitioning</i> to <i>In case of Partitioning</i> . Changing the options " <i>Available</i> " and " <i>Not available</i> " to " <i>Consistent</i> ", " <i>Available</i> " and " <i>Turnable</i> "

---

FR-12(L)	Adapting colour of properties' captions in <i>Step2: Trade-Offs</i>	The background colour of the properties' captions shall be changed from yellow to a light blue, that fits in the context of the entire application.
FR-13()	Change of the colour scheme of the entire application	Instead of the dominating colours black, white and grey to shadows of blue.
FR-14()	Integration of existing example projects	Projects that exist within the prototypes of previous work shall be also included in the integrated version.
FR-15()	Change of copyright	The copyright after the integration process shall be <i>Apache License Version 2.0</i> .
FR-16(L)	Integration of tab-mechanism for <i>Adapt Data Access Layer and Upper Application Layers</i>	The tabs with the advices for adaptations of migrated application's upper layers shall be in the integrated version.
FR-17 ()	Establish a tab-mechanism for the entire migration support.	Every migration step shall be in its own tab. If a step, like <i>step 1</i> , has several tasks to perform that tasks shall be subdivided as well.
FR-18 ()	Implement feedback loops.	If any conflicts are identified in <i>step 5</i> the user must have an easy way to switch to their possible causes.

---

**Table 4.2.:** Functional Requirements for CDMT After Integration

## 4.2. Add-on Extension Mechanism

In this section we specify the functional requirements of the add-on loading mechanism. To the contrary of the previous section (see Sect. 4.1) we have to develop this functionality from scratch. Hence, we will go more into detail to give a deeper understanding of this functionality. We describe the desired behaviour by *Use Cases*.

We subdivided this section by the major functions of the mechanism to provide a better overview. In the following we will refer to an add-on as *adapter* since adapter for database connectivity is the only type which will be provided by this mechanism.

In general we have to provide interfaces for the source and target adapters, which shall be included during runtime in CDMT. We also must extend the data model of CDMT since user shall be enabled to decide whether the adapter they provide might be used by others. Figure 4.1 illustrates use cases of CDMT. We provided the highlighted functionalities as part of this work.

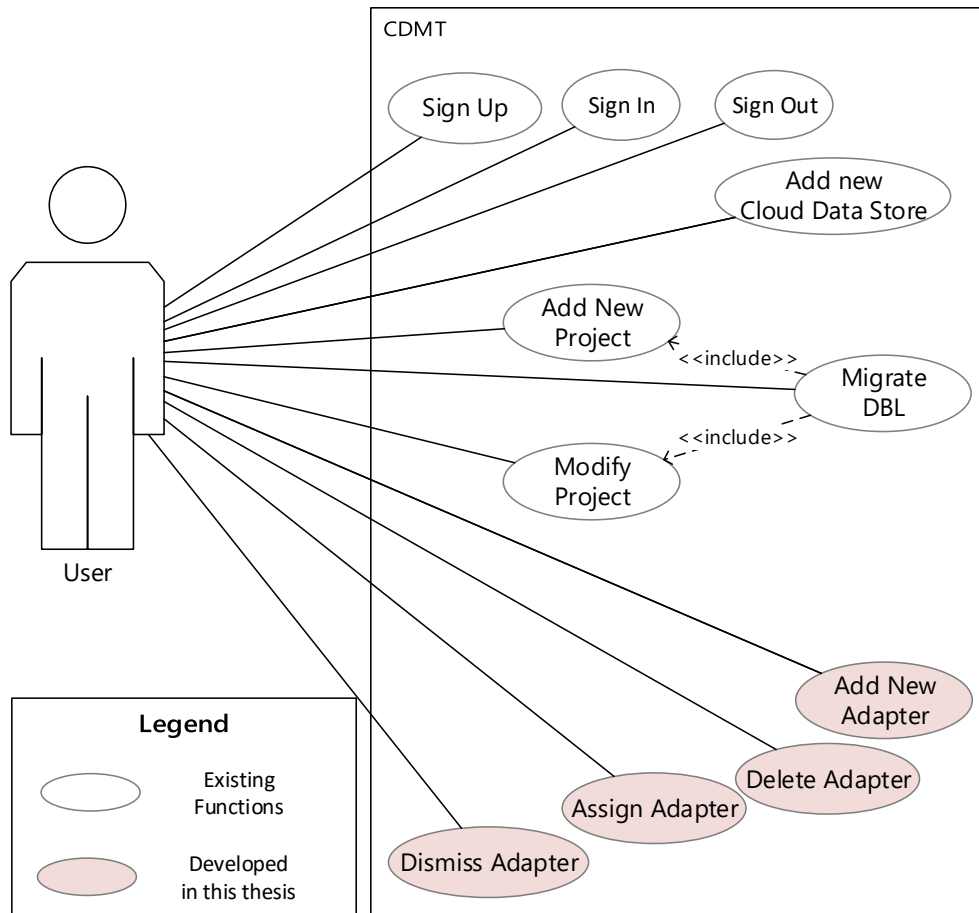


Figure 4.1.: Use Cases Overview

## 4.2.1. Add Adapter

<b>Name:</b>	Add Adapter
<b>Goal:</b>	The user wants to add a new database connectivity adapter to the list of available adapters.
<b>Actors:</b>	User, CDMT
<b>Pre-Condition:</b>	The user is registered, has signed in with his credentials, and selected the menu <i>Adapters</i> .
<b>Post-Condition:</b>	The selected adapter is stored in the data base. It is shown in the adapters list and is available. The archive file containing the adapter is copied to CDMT's adapter directory.
<b>Exceptional Post-Condition:</b>	The selected adapter is not stored in the data base and is not shown in the list of available connectivity adapters.
<b>Regular Case:</b>	<ol style="list-style-type: none"> <li>1. The user clicks on the <i>Add Adapter</i> Button.</li> <li>2. The CDMT opens the <i>New Adapter View</i>.</li> <li>3. The user fills in the required properties of the adapter.</li> <li>4. The user uploads the archive of the adapter.</li> <li>5. CDMT copies the archive of the adapter to its plug-in directory.</li> <li>6. The user acknowledges it by clicking on the <i>Confirm Button</i>.</li> <li>7. The CDMT writes an entry into the database with adapter's id and properties.</li> <li>8. The adapter is in the list of available connectivity adapters.</li> </ol>
<b>Exceptional Cases:</b>	<ol style="list-style-type: none"> <li>4a. The adapter has not the file *.JAR. <ol style="list-style-type: none"> <li>a) The upload function is disabled</li> <li>b) The user can either provide a proper file or aboard.</li> </ol> </li> <li>5a. The adapter cannot be accessed. <ol style="list-style-type: none"> <li>a) The user gets an error message about the failure.</li> </ol> </li> <li>7a. The database cannot be accessed. <ol style="list-style-type: none"> <li>a) The user gets an error message about the existing adapter.</li> <li>b) CDMT aborts the operation.</li> </ol> </li> <li>7b. An adapter with similar properties is already added. <ol style="list-style-type: none"> <li>a) The user gets a message about the existing adapter.</li> <li>b) The user must acknowledge this message.</li> </ol> </li> </ol>

Table 4.3.: Use Case Add Adapter



### 4.2.2. Delete Adapter

<b>Name:</b>	Delete Adapter
<b>Goal:</b>	The user wants to remove a connectivity adapter from the list of available adapters.
<b>Actors:</b>	User, CDMT
<b>Pre-Condition:</b>	The user is registered, has signed in with his credentials and selected the menu <i>Adapters</i> . An adapter was added previously to the list, is visible, and accessible.
<b>Post-Condition:</b>	The deleted adapter is not shown in the list of available connectivity adapters or is accessible in any other way. The archive file of the adapter was deleted from CDMT's directory for adapters.
<b>Exceptional Post-Condition:</b>	The adapter is still accessible and is shown in the list of available connectivity adapters.
<b>Regular Case:</b>	<ol style="list-style-type: none"> <li>1. The user selects the adapter he/she wants to delete.</li> <li>2. The system requires a confirmation, if the action should be taken.</li> <li>3. The user confirms the action.</li> <li>4. CDMT removes adapter's specific entry from the database.</li> <li>5. CDMT deletes adapter from its plug-in directory.</li> <li>6. The added adapter is removed from the list of available connectivity adapters.</li> </ol>
<b>Exceptional Cases:</b>	<ol style="list-style-type: none"> <li>4a The selected adapter cannot be deleted since it is still assigned.               <ol style="list-style-type: none"> <li>a) The system shows an error message.</li> </ol> </li> <li>5a The selected adapter does not exist.               <ol style="list-style-type: none"> <li>a) The system shows a message.</li> <li>b) Continuing with regular procedure.</li> </ol> </li> <li>5b Adapter's file cannot be accessed.               <ol style="list-style-type: none"> <li>a) CDMT shows an error message.</li> </ol> </li> </ol>

**Table 4.4.:** Use Case Delete Adapter

## 4.2.3. Assign Adapter

<b>Name:</b>	Assign Adapter
<b>Goal:</b>	The user wants to use an available connectivity adapter.
<b>Actors:</b>	User, CDMT
<b>Pre-Condition:</b>	The user is registered, has signed in with his credentials and selected the menu <i>Adapters</i> . An adapter was added in the list of available connectivity adapters previously. It is visible and accessible.
<b>Post-Condition:</b>	The highlighting of adapter's <i>Assign-Button</i> is changed from green to purple, its label shows the entry <i>Dismiss</i> instead of <i>Assign</i> . The Adapter is usable to establish connections.
<b>Exceptional Post-Condition:</b>	The state of adapters button is not changed and the adapter cannot be used to establish connections.
<b>Regular Case:</b>	<ol style="list-style-type: none"> <li>1. The User clicks on the <i>Assign-Button</i> of the desired adapter from the list of available connectivity adapters.</li> <li>2. CDMT reads out the properties of the selected adapter.</li> <li>3. CDMT adds the adapter to system's class loader.</li> <li>4. CDMT adds the user assignment of the adapter to the database.</li> <li>5. The <i>Assign-Button</i> of assigned adapter changes the colour to purple and shows the label <i>Dismiss</i>.</li> </ol>
<b>Exceptional Cases:</b>	<ol style="list-style-type: none"> <li>2a. The user does not have the permission for that adapter anymore. <ol style="list-style-type: none"> <li>a) CDMT shows an error message.</li> <li>b) CDMT removes the respective adapter from the list of available connectivity adapters.</li> </ol> </li> <li>3a. The database cannot be accessed. <ol style="list-style-type: none"> <li>a) CDMT shows an error message.</li> <li>b) CDMT removes the adapter from system's class loader.</li> </ol> </li> <li>5a. The adapter does is not implemented correctly. <ol style="list-style-type: none"> <li>a) CDMT shows an error message.</li> <li>b) CDMT removes adapter specific entries from the database.</li> <li>c) CDMT removes adapter from its plug-in directory.</li> </ol> </li> </ol>

Table 4.5.: Use Case Assign Adapter

#### 4.2.4. Dismiss Adapter

<b>Name:</b>	Dismiss Adapter
<b>Goal:</b>	The user does not want to use an connectivity adapter anymore.
<b>Actors:</b>	User, CDMT
<b>Pre-Condition:</b>	The user is registered, has signed in with his credentials and selected the menu <i>Adapters</i> . An adapter was previously assigned from the list of available connectivity adapters. It is visible and accessible.
<b>Post-Condition:</b>	The <i>Assign-Button</i> of the adapter changes its colour from purple to green, and the its label is changed to <i>Assign</i> . User's assignment is removed from the database.
<b>Exceptional Post-Condition:</b>	The adapter stays assigned and visible in the list of user's available adapters. The <i>Assign-Button</i> remains purple and its label was not changed.
<b>Regular Case:</b>	<ol style="list-style-type: none"> <li>1. User clicks on the purple <i>Assign-Button</i> of the adapter which he wants to dismiss.</li> <li>2. CDMT removes the assignment entry from the database.</li> <li>3. CDMT removes the adapter from system's class loader.</li> <li>4. The <i>Assignment-Button</i> of the adapter is changes its colour to green and its label displays <i>Assign</i>.</li> </ol>
<b>Exceptional Cases:</b>	<ol style="list-style-type: none"> <li>2a. The database cannot be accessed. <ol style="list-style-type: none"> <li>a) CDMT shows an error message.</li> <li>b) CDMT aborts the operation.</li> </ol> </li> <li>3a. The adapter cannot be removed from system's class loader. <ol style="list-style-type: none"> <li>a) CDMT shows an error message.</li> <li>b) CDMT aborts the operation.</li> </ol> </li> </ol>

**Table 4.6.:** Use Case Dismiss Adapter

### **4.3. Non-Functional Requirements**

In this section we describe the most important non-functional requirements towards our implementation. We exclude applications, whose DBL is migrated with our application, from meeting this requirements, since every developer on his own is responsible for his work.

#### **4.3.1. Extensability**

Nowadays there are plenty of data hosting solutions, but in quite short periods more provider appear on the market. To ensure utility of the CDMT we have to provide an opportunity extending it to be able migrating data to such new providers.

#### **4.3.2. Usability**

An important aspect is to afford a understandable and intuitive way to use the CDMT. It has to guide the user through the migration scenario in an understandable and reconstructing manner. The UI has to be recognizable to ensure that new users, or users who haven't been using it for a certain period of time, can manage the usage without unpleasant obstacles or an extensive effort to learn the concept of the tool.

#### **4.3.3. Reuseability**

The source code of the CDMT has to be accessible to developers beyond the period of our implementation. We attempt to deliver an architecture which allows to be adapted to further use. To eliminate legal concerns a license which allows an ongoing development of the CDMT shall be used.

#### **4.3.4. Integratability**

The CDMT has to provide a utility without having exorbitant and unnecessary impact on the application which DBL is to migrate. Furthermore, it has to be integrable in migration projects without the demand of changing the strategy of the process or the functionality of the environment.

#### **4.3.5. Maintainability**

Providing a high degree of maintainability has to be reached by an understandable implementation. For further comprehension of the source code implemented methods and modules have to be documented by comments explaining the purpose of the respective parts.

#### **4.3.6. Backward Compatibility**

The implementation of the functional requirements specified in Section 4.1 and in Section 4.2 must not have any impact on the projects which were created in the previous versions of the CDMT. Furthermore, they must stay usable and configurable at the end of the integration process, as well as after the implementation of further functionality.

#### **4.3.7. Security**

The data belonging to a certain user must be protected in the way that no others without permission are able to access it. Furthermore changes in the source code of CDMT shall be done in a way that prevents malicious manipulations.

#### **4.3.8. Portability**

The system shall be independent from the environment. Furthermore, as little as possible the system shall be dependent on installations an additional products in the runtime environment after it is build an distributed.



---

## 5. Design

---

In this chapter we introduce the architectural solution and improvements we designed to fulfil the requirements presented in Chapter 4. We explain the redesigned workflow for the decision support of CDMT in Section 5.1, as well as the changes we made in the implementation, after the first iteration. Followed by Section 5.2 in which we present the major innovations of the architecture.

### 5.1. Adaptations

In this section we present the necessary adaptations to fulfil requirements from Chapter 4. It is subdivided in two main sections. We introduce changes we made in the presentation layer as part of the integration process in Section 5.1.1. As result of this changes it was possible to adapt the workflow of the decision support to new requirements which we present in Section 5.1.2.

#### 5.1.1. Presentation Layer

To provide a high degree of maintainability (see Sect. 4.3.5) and extensibility (see Sect. 4.3.1) we redesigned the structure of the presentation layer. We split the presentation of the decision support in multiple components as depicted in Figure 5.1. This design allows to exchange or manipulate the view of every step separately and easily to extend the Web UI of the CDMT. All the components are in the same folder, hence we illustrated it as a *Fundamental Modeling Concepts (FMC) block diagram* to be able to explain the connections between respective components implemented as JSPs. Every single JSP contains an HTML definition to generate the content and method calls of the respective CDMT's Web UI. All the pages are embedded within the `migration-settings.jsp` which loads the respective page for user's selection. Pages surrounded by a dashed boarder with round corners are varying content, which is loaded in dependence on the selected migration scenario. The page `get_project-settings-overview.jsp` gives an overview of set properties in the consecutive steps of the decision support.

Furthermore, it allows to call the needed page directly if properties selected in one of the steps shall be changed. Pages which allow an active selection are characterised with a prefixed `set_`. Some pages, as `adapt-upper-app-layer.jsp`, contain advices for the user how he has to adapt his application to be able using the migrated DBL. Another important page which does not provide direct settings is `identify-solution-patterns.jsp`. Its purpose is to identify possible conflicts of properties set for the migration and to suggest patterns to resolve these by changing properties in the linked steps which are responsible for the target store definition (see Fig. 5.1).

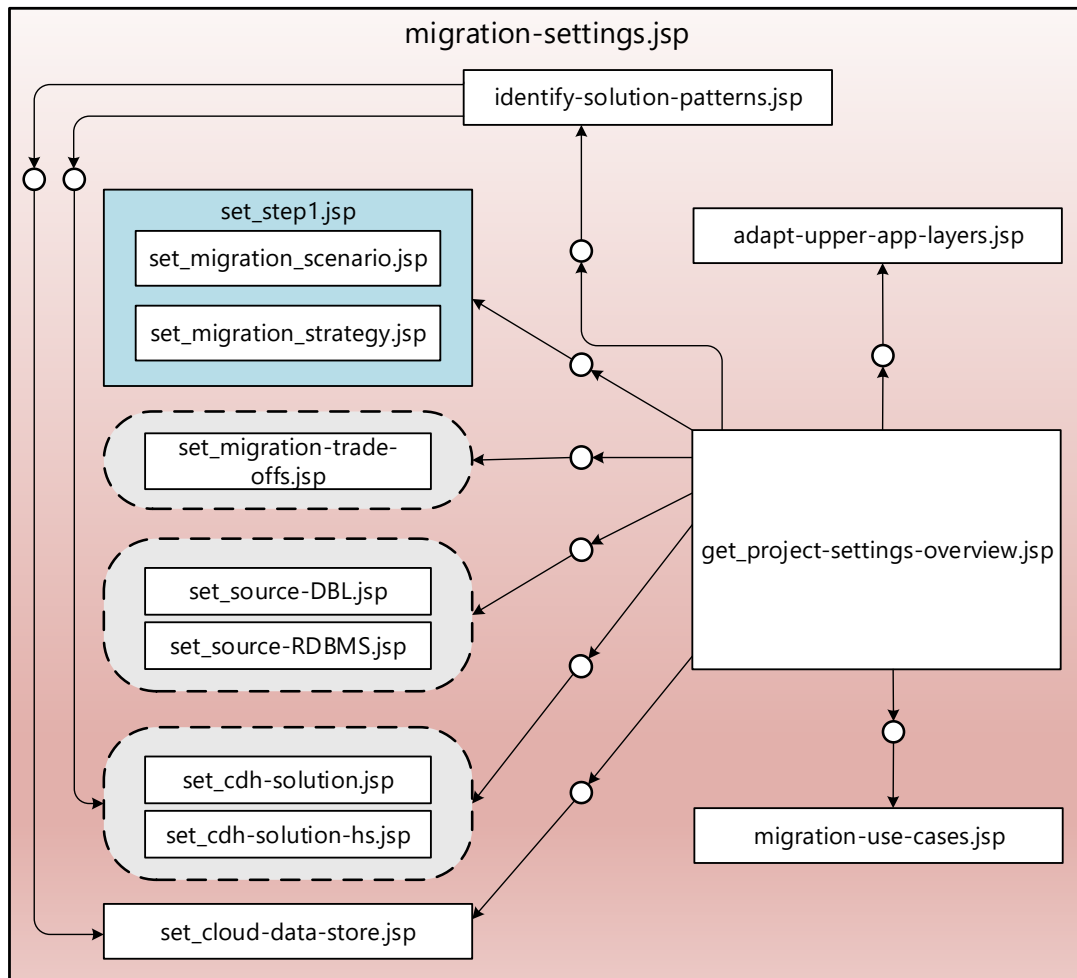


Figure 5.1.: Component Connection of the Presentation Layer

### 5.1.2. Workflow

In this sections we will introduce the workflow of a migration process with the CDMT. We depicted it as a simplified flow chart in Figure 5.2. Every process in this diagram represents a major step of the migration process. The workflow also correlates with the structure of the presentation layer's components we explained in Section 5.1.1. In the first step *Set up Migration Scenario* the User selects the desired scenario and refines it. If this scenario has the purpose to migrate from a RDBMS to a NoSQL store the user has to *Identify Trade-Offs* regarding his migration in the second step, else he continues with the description of his source data store in the the third step *Describe Source Store*. Independently of the scenario choice step two will be displayed in the Web UI, but is not configurable for scenarios other than *RDBMS to NoSQL*. The difference between the steps *Describe Source RDBMS* and *Describe Source Store* is that the first one mentioned is more restrictive since the scenario prescribes the source store



## 5.1. Adaptations

---

as a RDBMS. After accomplishing the third step between the scenarios distinguishes do not matter and the workflow continues with the same consecutive steps for every scenario. In the fourth step *Describe Desired Target Store* the user has the choice to set various properties which his target desired store should support. Afterwards the user eventually selects one of the predefined data hosting solutions in the fifth step *Select Target Store*. At this point the definition of the data migration is almost finished in the sixth step *Identify possible Conflicts* a list along the respective conflicts might occur between the previous selections is given. If that should be the case one can get to the steps where the properties of the target store and the selected data hosting solution are set. This mechanism supports a feedback loop to minimize side effects on the data due to the migration process. Hopefully as all conflicts are resolved the user gets in the seventh step *Adapt Upper Application Layer* multiple advices to adapt his application for a proper data access in the environment where the data is migrated. Once all the preparation is done the DBL finally will be migrated to the desired environment in the last step *Migrate Data*. After this step is executed the migration process terminates.

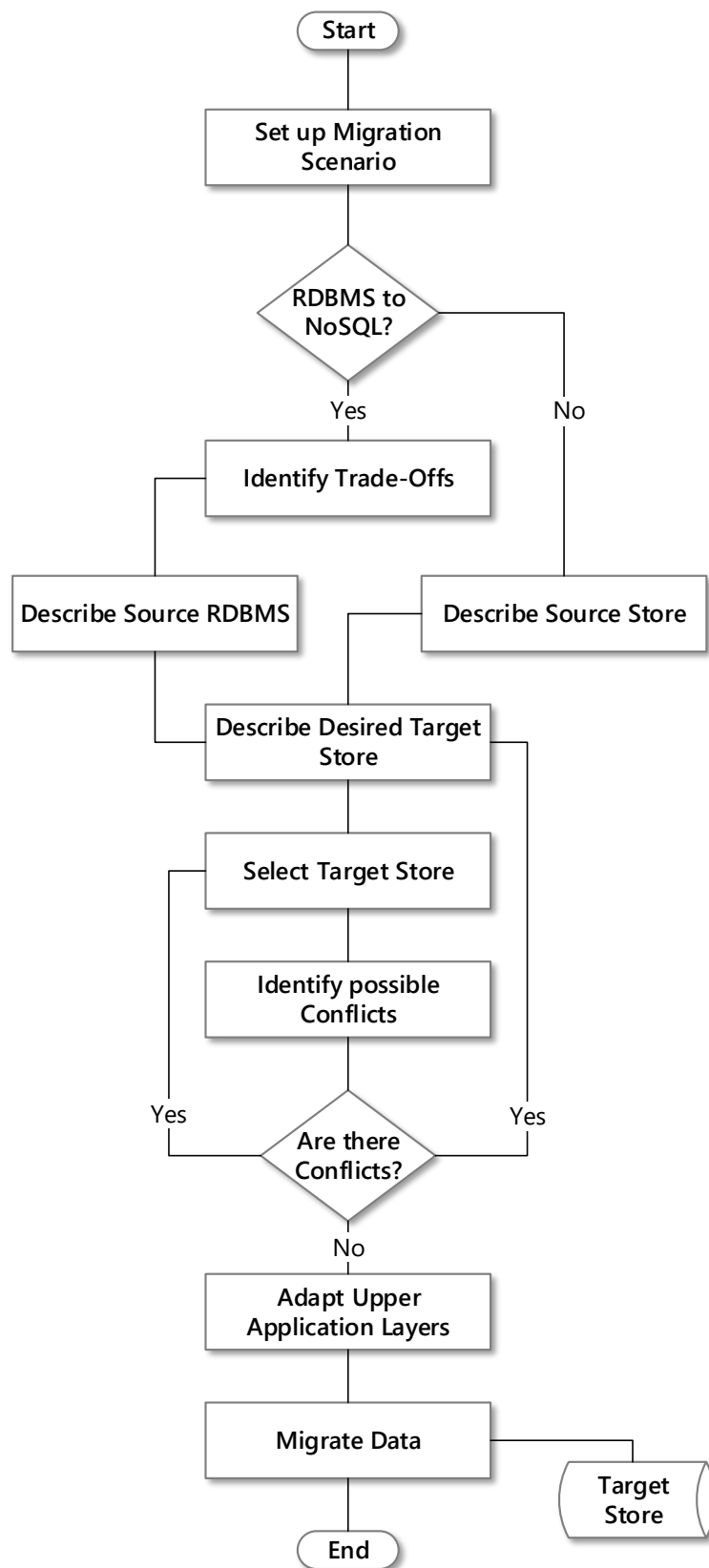


Figure 5.2.: Workflow for the Decision Support

## 5.2. Extension Mechanism Architectural Overview

In this section we present our design to meet the requirements in respect to provide extensions (see Sect. 4.2). In Section 5.2.1 we introduce the design and architecture which enables loading add-ons during runtime. Further extensions were made in the schema of used MySQL DB which we highlight in Section 5.2.2.

We also considered the non-functional requirements while designing the new functionalities.

### 5.2.1. Architecture of the Extension Mechanism

In this section we explain how the designed extension mechanism for the add-on functionality works. We introduce that mechanism isolated from the existing software environment for the sake of simplicity and a better comprehension.

The extension mechanism requires several new software components for a proper functionality. Furthermore, we stick to the layered architecture, originally initiated by Bachmann [Bac12] (see Sect. 2.3.2), to have a consistent architecture and a certain degree of maintainability. The components are embedded in the respective layer as Figure 5.3 depicts. We provide a certain degree of security (see Sect. 4.3.7) by copying the archive files that contain a selected adapter to CDMT's adapter directory. That way we cumber malicious manipulation of data which might lead to a severe exception.

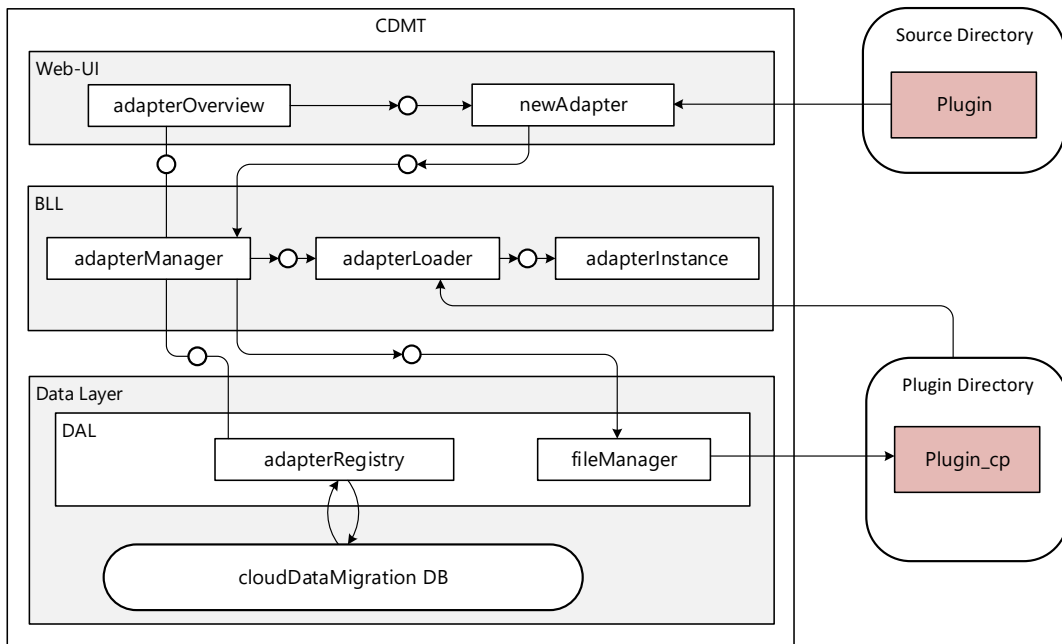


Figure 5.3.: Add-on Mechanism Overview

We introduce the components and their scope layer by layer to reveal their functionality in the same chronological order as their invocation during usage below. Moreover, we explain for every component which other components it accesses to provide a better comprehension of Figure 5.3. The anchor point of all operations, which this mechanism delivers, is the component *manageAdapter*.

We begin with the most upper layer, *Web UI*. Its purpose is to provide a graphical interface to trigger provided operations. The following list gives an overview of components which we used to extend:

- **manageAdapter:**

This component contains a view where the user is able to assign, dismiss, or delete adapters. It also contains the possibility to add a new adapter, whereas for this the component *addAdapter* is responsible.

In any other case the component *adapterManager*.

- **addAdapter**

This component is invoked, if the user started the operation *Add Adapter* in the view of *manageAdapter*. It contains a view with a form where the user enters information that is mandatory to load a connectivity adapter. It accesses the component *adapterManager*

The next layer is the *BLL* which purpose is to communicate between the presentation layer and the resources which in this case are commonly supported by the Data Layer. We extend the *BLL* by the following components:

- **adapterManager:**

The purpose of this component is to direct operation requests from the component *manageAdapter* to components in the DAL and further components in the *BLL* itself. It depends on the executed operation which components are involved. If a new Adapter is added or removed the components *adapterRegistry* is involved and the *fileManager*. If the operation assign or dismiss are triggered, the components *adapterRegistry* and *adapterLoader* will be used for further processing.

- **adapterLoader:**

The focus of this component are two functions. The first is to add adapters to system's class loader that shall be used. The second one is to remove adapters from system's class loader which are not required anymore. It accesses the component *adapterInstance*. After an adapter is successfully added this component provides input to create an instance of the component *adapterInstance*.

- **adapterInstance** This component enables to create an instance of adapters which are added to system's class loader during runtime. It enables access to the methods and functionality of instantiated adapters.

The lowest layer is the *Data Layer*, which basically contains two sub-layers, the DAL and the DBL. The adaptations of the DBL that we make are explained in Section 5.2.2. The list below shows components that we add to the DAL:

## 5.2. Extension Mechanism Architectural Overview

- **adapterRegistry:** The responsibility of this component is to administrate adapters which are added. Furthermore, it updates the DB with information of every single adapter and if, and in which way a certain adapter is used. It accesses the resources from the DB and delivers their content to *adapterManager*.
- **fileManager** This component manages the physical files of CDMT's connectivity adapters. If an adapter is added this component copies the specified file into CDMT's adapter directory. As soon as an adapter is not longer required its file is deleted in the *Adapter Directory*.

### 5.2.2. DB Schema Extension

In this section we introduce the adaptations of the DB Schema we make to provide the extension mechanism for CDMT. Since the DB connectivity is still provided by adapters which are included in CDMT's source code there was no necessity to manage their properties additionally in a persistent manner. We extend the DB Schema by two additional tables which are highlighted in Figure 5.4.



Figure 5.4.: Snipped of Extended DB Schema

The table *AdapterRegistry* contains the user and the adapters which they are using. The id of an adapter is added in a row with the id of the user who assigned it. The more complex table

*Adapter* contains all information of adapters which are mandatory to enable the usage of an adapter in the specified way. For example we have the attribute *owner*, which contains the id of the user who added a respective adapter. This attribute is required due to the possible restriction of *private* usage only. If this restriction is set only the user who added that adapter is able to assign and use it.

---

## 6. Implementation

---

In this chapter we present how we realized the requirements from Chapter 4 with the design from Chapter 5. We divide this chapter into several sections in order to separate each stage of the implementation. We distinguish the content by the criteria if we extend the BLL or are able to accomplish tasks without changing the base model of CDMT. In Section 6.1 we explain how we accomplished the integration phase. In this section we do not change the BLL or the data model of CDMT. Afterwards we migrated the entire project into a new environment, we reveal the necessity and benefit of this approach in Section 6.2. Accordingly we leverage the benefits of the migration to adapt CDMT towards new the requirements and introduce the innovations in Section 6.3. In Section 6.4 we give an overview of new features we implement, that increase the usage of CDMT and required an extension of the data model. Eventually we finish this chapter with Section 6.5 in which we outline a source and a target adapter for the sake of completeness. Furthermore, we require theses adapters to validate our implementation (see Sect. 7).

### 6.1. Implementation of the Integration

In this section we describe the implementation of the integration phase. We subdivide this topic in two further sections. First of all we introduce the tools and libraries we used for this iteration in Section 6.1.1. Section 6.1.2 gives an overview of the components we integrated in the base version and the tools we used that supported this process. Additionally we explain how we reused and rearranged some fragments of the previous implementations.

#### 6.1.1. Tools and Libraries Used for Integration

In this section we give an overview of the tools we used during the integration process. A brief description of the respective tools help to identify its scope.

**Apache Tomcat**<sup>1</sup> (Version: 7.0.26)  
Servlet container to publish and run CDMT.

---

<sup>1</sup><http://tomcat.apache.org>

**Eclipse<sup>2</sup>** (Version: 3.7 "indigo")

The *Integrated Development Environment* (IDE) where CDMT was initially created and edited. Its plug-in based concept allows to extend its functionality.

**Java** (Version: 1.6.0\_24)

Java is a programming language and compiler which are used for the development of CDMT's business logic and data access layer.

**JQuery<sup>3</sup>** (Version: 1.9.1)

JQuery is a library for *JavaScript*<sup>4</sup> (JS) that enables building and inspecting a *Domain Object Model* (DOM) of a HTML page. Furthermore, it provides various methods to access and manipulate specific objects in the DOM, which are represented as HTML nodes.

**JQuery UI<sup>5</sup>** (Version: 1.10.3)

JQuery UI is build on top of JQuery JS Library. It contains a set of *Cascading Style Sheets* (CSS) to design Web pages, and adapted JQuery functionality to provide a defined user interaction with Web-UI elements.

**Meld<sup>6</sup>** (Version: 1.5.3)

Editor to compare content of directories and files of the concurrently developed versions of CDMT.

**Oracle MySQL Workbench<sup>7</sup>** (Version: 5.2.38 rev. 8753)

Graphical tool to access and adapt the DBL of CDMT.

**Oracle VM VirtualBox<sup>8</sup>** (Version: 5.0.14 rev. 105127)

VirtualBox is a virtual machine manager to host the system which contains the development and execution environment as well es the source code of CDMT. It is used to keep an image of a stable configuration and encapsulate the development in a save environment.

---

<sup>2</sup><https://eclipse.org>

<sup>3</sup><https://jquery.com>

<sup>4</sup><https://www.javascript.com>

<sup>5</sup><https://jqueryui.com>

<sup>6</sup><http://meldmerge.org>

<sup>7</sup><https://dev.mysql.com/downloads/workbench>

<sup>8</sup><https://www.virtualbox.org>



### 6.1.2. Integration Strategy

In this section we explain our approach to integrate components of the previous implementation that were missing in our base version. Furthermore, we give an introduction of the steps we had to complete and the included actions we performed as part of every step. That after we give an overview of the components which were contained in only one of the versions we integrated.

#### 1. Identification

The first task was to identify intersection of components and functionality between the versions, which we have done by respective comparing the project directories. For this purpose we used *Meld*. Afterwards all distinguishing components were extracted into a separate folder. Following this, it was necessary to get all differences between files and components which once had the same implementations but were adapted as part of the respective work. This task required a manual inspection of the supplementary source code parts to decide which dependencies and impact it would have after the integrations. As soon as these tasks were finished we went on with the step Inclusion.

#### 2. Inclusion

This step required a raw integration of the distinguishing components. We used the import function of the IDE *Eclipse* to include the additional components in our development base. Afterwards we had to include external Java libraries which were used by some classes. Support for the identification of such libraries was provided by the syntax analysis of *Eclipse*. After we had included the necessary files we continued with the step Adaptation.

#### 3. Adaptation

To provide a proper use of the added component some source code files had to be adjusted. For this task we used the information gained in the step Identification to localize the files. Further aid was given by *Eclipse* since all components that were not used were highlighted in the project structure. For example we had to create a new instance of the included data store adaptors within the class `CloudDataMigrationContextListener.java` to be able to use them.

Furthermore, we had to change the chronological order of steps which are performed while a migration with the CDMT. We customised the file `project.jsp` to fulfil *FR-3(L)*, *FR-3(L)*, *FR-5(L)* and *FR-16(L)* (see Sect. 4.1.2).

We had to execute certain SQL statements with *Oracle MySQL Workbench* to adapt the data model in the DBL. We extended the methodology by additional entries to accomplish requirements from *FR-4(L)* to *FR-11(L)* as well as *FR-1(L)*, *FR-2(L)* and *FR-14()*. After we had solved all conflicts and provided the necessary dependencies we could proceed with the step Validation.

#### 4. Validation of Basic Functions

This step has just few requirements. The approval of a correct system behaviour is part of Chapter 7. In this step we inspected the following criteria of the CDMT:

- **Bootability:**  
Does Apache Tomcat publish the application after the performed integration tasks?
- **Previous DB Entries:**  
Are previous entries in the DB like user credentials or created projects visible and accessible?
- **Adapted Workflow:**  
Are the steps of migration displayed and executable in the changed order?
- **Extended Methodology:**  
Are added categories and properties visible and selectable?
- **Behaviour of the Web UI:**  
Are the Web pages presented in the same order as before the the integration tasks were performed?

After we ensured the all requirements listed above were satisfied we considered this part of the integration as successful.

## 6.2. Migration and Reconstruction

In this section we describe the migration of the CDMT project into a new environment as well as the new project configuration and build process. We subdivide this section into further subsections to emphasis the benefits of the respective changes. In Section 6.2.1 we explain the motivation which lead us to migrate and reconstruct the entire development project. Following this, we briefly introduce the new tools we use for the different stages of the software life cycle in Section 6.2.2. Afterwards, we present the improved build cycle in Section 6.2.3. Finally, we cover the deployment of CDMT and its the significant changes in Section 6.2.4.

### 6.2.1. Motivation

We were forced to perform this task by the state of CDMT's development project. After we finished the integration we have to install several new development tools in order to extend CDMT with appropriate technologies and practices, since the development environment that was used before is outdated and provides barely support. The first attempt is to create a repository with *Github*<sup>9</sup> to clone the project in an environment with suitable equipment. This leads to a dead end since several dependencies were neither documented nor obvious. To enable better development support in order to reduce potential failures and workarounds, the entire project was reconstructed. A part of this reconstruction was to clarify which external libraries are used in which phase of the build cycle and which are not necessary or obsolete. Furthermore, a major concern is to provide extensibility (see Sect. 4.3.1), so that future changes can be done without obstacles for developers and reduce dependence on software products. Additionally we improve the maintainability (see Sect. 4.3.5) since libraries can be exchanged without having to research the entire source code.

### 6.2.2. Extended Toolset

As a result we are able to include further tools to accomplish the implementation more efficiently. Below we introduce the tools we used to improve the development process.

#### **Apache Maven**<sup>10</sup> (Version: 3.3.3)

We use Maven as build servers. It supports and encourages usage of best practices. Since the whole project dependencies are declared in a *Project Object Model* (POM) file the dependencies are transparent. Furthermore, it provides a repository with plenty of Java libraries and plugins to use for a build project. Additionally we will leverage the fact that unit test are a part of the build process, so that we are able to perform an integration process easily.

#### **Docker Engine**<sup>11</sup> (Version: 1.11.1)

Docker is a open source project on Github. The Docker Engine provides a lightweight runtime engine. An in-host daemon communicates with the Docker client to support functionalities as building, shipping, and running a Docker container. A great benefit of containers is the encapsulation of an application and its dependencies. Since it shares the kernel with the host system an efficient resource management is the result and by running as process in separate userspace the host system can be prevented from endangerments caused by application failures. Additionally container do not require a specific infrastructure and might be deployed on various environments.

---

<sup>9</sup><https://github.com>

<sup>10</sup><https://maven.apache.org>

<sup>11</sup><https://www.docker.com/products/docker-engine>

**Jetbrains IntelliJ IDEA Community Edition<sup>12</sup>** (Version 2016.1.1)

IntelliJ IDEA is an IDE for Java projects with a great amount of built-in tools and frameworks. Primarily we decided to use it as IDE since it provides a performant debugger with remote functionality. Furthermore, we did not had to research required plug-ins to resume development of our project, since everything is included.

**Microsoft Visual Studio Code<sup>13</sup>** (Version: 0.10.11)

*Visual Studio* (VS) Code is a lightweight editor for various programming languages. It provides Git integration as default. We choose that tool as it has a large possibility to integrate plug-ins for Web-UI development such as *Bootstrap*<sup>14</sup> (BS). We use it to adapt and develop all Web-UI related components of CDMT.

### 6.2.3. Build Cycle

In this subsection we outline the build phases of *Maven's* build life cycle we configure and the plug-ins we use for our purpose. We do not explain the functionality of *Maven* or its build life cycles, for those who want to know more about this topics we suggest [apc] and [jav] to obtain required information. We describe each build phase, including the plug-ins used for the respective phase as well as its purposes and coherence for our attempt below. It is important to mention that the build phases depend consecutively on each other. Besides the goals that are defined for every execution in a phase are functionalities which are provided by referenced the respective plug-ins referenced in a phase.

#### Process-Classes

In this phase we use the *Datanucleus*<sup>15</sup> *Maven Plug-In* which goal is *enhance*. We have to use this functionality in order to enable Tomcat to use CDMT's classes. Otherwise, we get errors in context of persistent Java objects.

#### Process-Test-Sources

In this phase we use the *Surefire*<sup>16</sup> *Maven Plugin* in order to execute JUnit<sup>17</sup> tests with each build. The goal of this phase is *test*. We need this build phase to ease the integration of new components.

---

<sup>12</sup><https://www.jetbrains.com/idea>

<sup>13</sup><https://code.visualstudio.com>

<sup>14</sup><http://getbootstrap.com>

<sup>15</sup><http://www.datanucleus.org>

<sup>16</sup><http://maven.apache.org/components/surefire>

<sup>17</sup><http://junit.org>

### Pre-Integration-Test

In this phase we use the *fabric8*<sup>18</sup> *Maven Plug-in*. Its goals are *build*, *start*, and *stop* docker containers from a specified image, which is provided by Docker. We use this approach to provide an isolated user-space for CDMT.

### Generate-Resouces

We use *Maven Resources Plugin* in this phase, which copies the resources from a specified source directory to a defined output directory. This configuration is necessary due to different resources were already created in CDMT's development project. We use them as profiles of which each provides a configuration with example connection parameters for CDMT's DB.

### 6.2.4. Deployment

It is important to know how a software is deployed in order to be able to run it correctly. Additionally it is for maintenance engineers important to know in which way a productive system is deployed to include this phase in their projects. Figure 6.1 depicts the structure of deployed CDMT. Whereby the application and the DB have their own container each. The *App Container* is the environment of CDMT and contains all required dependencies to run it as well. Furthermore, it has installed Apache Tomcat to publish the application. The *DB Container* has installed *Oracle MySQL* in it and provides resources from the *CDMT DB* which is basically data, like user information, to run CDMT. The communication between the containers is managed by *Docker Engine* via virtual networks.

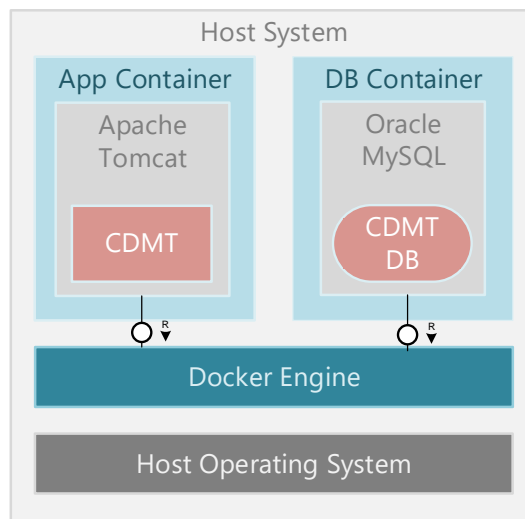


Figure 6.1.: Structure of Deployed System, inspired by [doc]

---

<sup>18</sup><http://fabric8.io>

We deploy CDMT in containers to protect the host machine, where development takes place, from errors caused by failures of CDMT. Furthermore, we benefit from the fact that we have a stable build with a defined DB state, which is provided by a configured dump file. However, if the DB should be corrupted by failures in the source code of the components which access the DB we are able to start from a stable state by executing the build life cycle once again. Additionally we supply a more realistic scenario for test by logically separating the application server, which publishes CDMT, and the client, which can be a browser on the development machine.

### 6.3. Adaptation of Migrated Project

Not only technical progress during with passing time is a major aspect of software, but new perceptions in the field of UI as well. To keep up with recent requirements we adapted the UI by implementing approved concepts of user control. We give an overview of newly introduced, or updated libraries in Subsection 6.3.1. In Subsection 6.3.2 we deliver a brief explanation of the improved user control.

#### 6.3.1. Libraries

To enable contemporary some updates are necessary. We list the external resources with the used version and a brief description below.

##### **JQuery** (Version: 1.12.3)

We use a newer version of JQuery for our implementation since it has been extended and provides a support by the Web-community.

##### **Bootstrap** (Version: 3.0.3)

Bootstrap provides an comprehensive definition for Web-UI elements which have a consistent design and are easy to use. Since it has many users and a detailed documentation it supplies a good support for developers and inspires to use appropriate UI controls.

##### **Bootstrap Paper**<sup>19</sup>

Bootstrap Paper is a custom theme with consistent design. It contains CSS, which provides the design, and an additional JS file, to have several design related functionalities such as a fully developed tab-mechanism. We use this component to design the entire Web-UI of CDMT. We chose this Bootstrap Paper since it fits well in the metaphor of forms to set properties of a migration project in CDMT. Furthermore, a clean and tidy look does not distract the user from the major task and eases obtaining of information supported by the Web-UI.

---

<sup>19</sup><https://bootswatch.com/paper>

### 6.3.2. User Control

In this section we present implemented improvement suggestions. Especially we will explain the implementation of the *tab-mechanism* quite detailed, since it has a great impact on the usage of the CDMT. Figure 6.2 shows the view after creating a new project, or opening an existed in CDMT. We adapted the operating concept to provide a better user guidance, which leads to a improved usability (see Sect. 4.3.2) in general. Furthermore, this concept fulfils *FR-17(N)*. This concept was a result of rearranging items within the presentation layer, as introduced in Section 5.1.1.

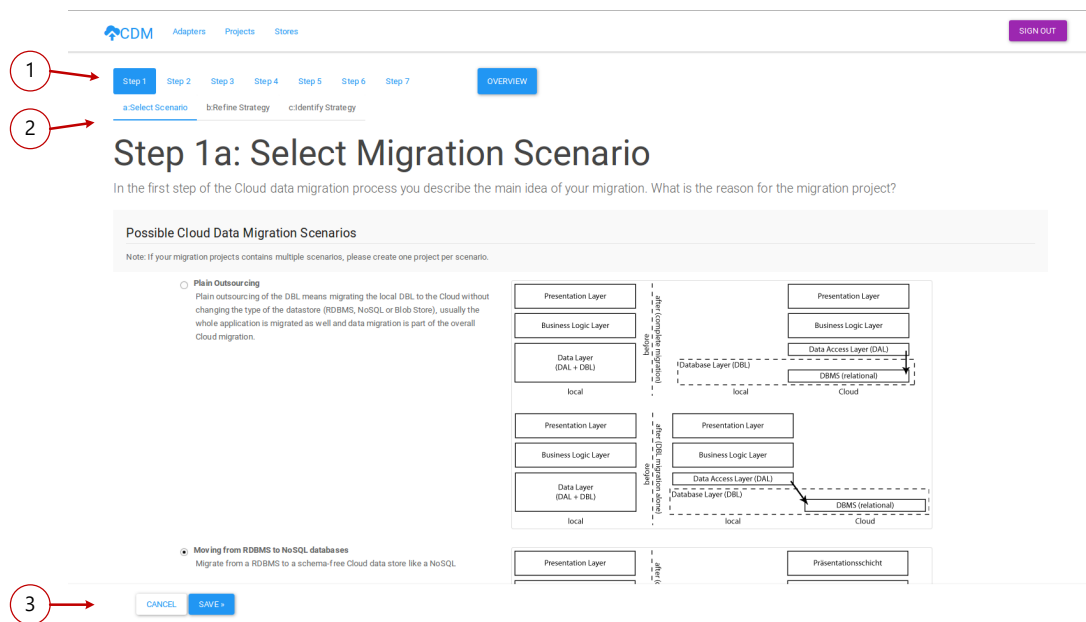


Figure 6.2.: Initiative View After Creating a new Project

Bellow we explain the concept of implemented changes. The points in the list below are referencing the numbered arrows in Figure 6.2.

#### 1. Navigation Tab Bar:

Each step of the migration methodology is divided in a separate tab. This way the user can always see in which stage of the migration he or she is and estimate the progress. Furthermore, it allows a comfortable switching between each step, if for example something should be changed.

#### 2. Task Bar:

As some steps in the methodology require more than one task. We subdivided such steps in tabs as well. The benefit is similar as the one of the *Navigation Tab Bar* but also gives a comprehensive order of the tasks and an overview what is to be done in such steps in advance.

#### 3. Bottom Bar:

Since some sheets of the migration methodology are quite long but have not necessarily

be viewed or filled completely we fixed the action buttons at page's bottom. For an example of this view, see Figure 6.2. The problem is that you can see not really much more than one scenario at the same time; so if you chose the first one it can be boring and cumbersome to scroll all the way down to the bottom of the sheet. By using our new and improved design you can progress your migration process immediately after selecting the desired option by clicking on the Button *Save*.

By the changes explained above we provided a more efficient Web UI. This way we reduced time consumption of mandatory operations without losing any information or skipping essential options.

## 6.4. Extension of CDMT

In this section we outline the implementation of the extension mechanism we designed in Section 5.2. We had to extend the data model in order to provide additional functionality. More precisely, we provide by accomplishing this task the possibility to upload and integrate connectivity adapters for every desired store during runtime. We begin with Subsection 6.4.1 to give an overview of the new structure of the Web-UI. In Subsection 6.4.2 we briefly introduce the *"Dashboard"* we integrated to CDMT. Afterwards we explain the implemented *"Adapter Overview"* in Subsection 6.4.3. Following this, we present the *"New Adapter Page"* in Subsection 6.4.4. We finish this section by providing a description of the concept we use to prove the correctness of our implementation and integration of the new components into CDMT in Subsection 6.4.5.

Generally we use AJAX messages to send request from the browser to the server and responses in the opposite direction, as shown in Figure 6.3. Since that is a common method to transfer messages. We will not further explain its functionality but recommend [w3s] for detailed explanation. This messages are responsible to obtain data for CDMT's Web-UI and initialize server operations by the client.

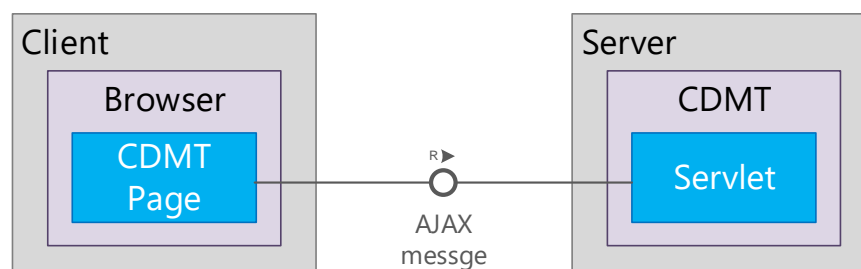


Figure 6.3.: Message Exchange Schema



### 6.4.1. Web-UI Navigation Controls

We begin with the entry point of the Web Application, the *Welcome Page* which is illustrated in Figure 6.4. Our motivation is to deliver a UI which has a clean appearance and intuitive design to allow the user focussing on the functionality without having to learn the controls.

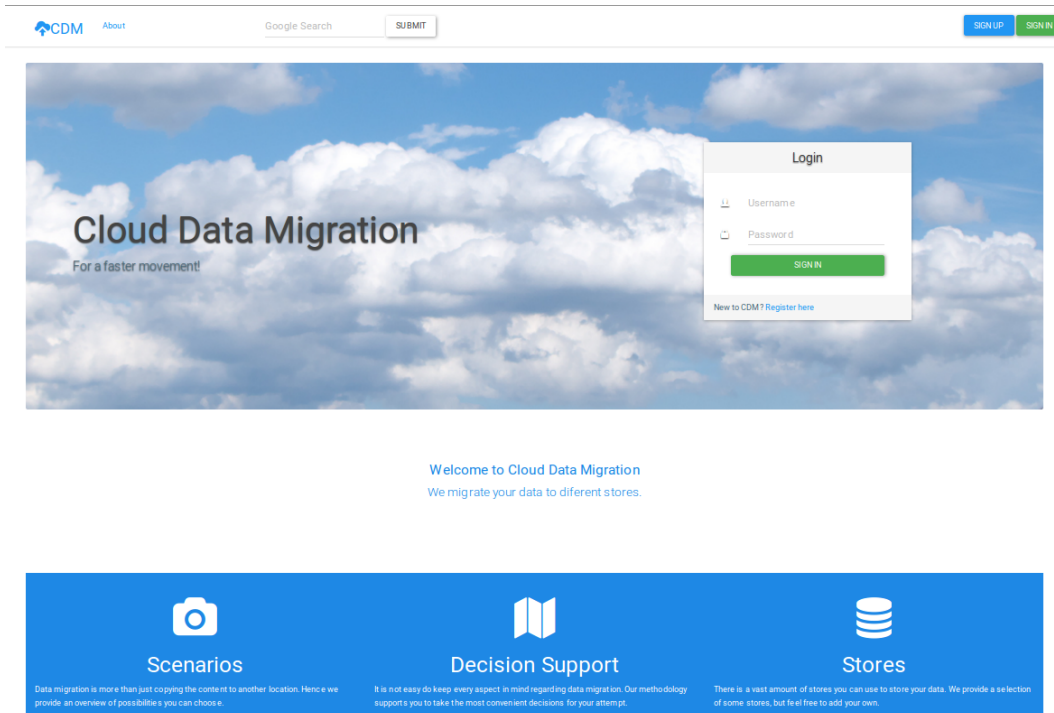


Figure 6.4.: Welcome Page

The most important thing to know about this page is, that the *Home Button* is in the upper left corner in the navigation bar. It will always call that page independently on which one you are in the *public area*. For further explanation on this page we move on to the general concept, since it clarifies the purpose of the *Welcome Page* as well. Figure 6.5 depicts the coherencies between respective Web pages.

The blue arrows in this illustration symbolize the connections between the pages and the direction from which page you are able to navigate to which other. Whereby the arrowhead points at the target, moreover the one with two arrow heads represent a bidirectional navigation.

The rectangles represent HTML pages, further referenced as "pages", that are contained in the Web-UI of CDMT. Furthermore, this picture categorises in three major types of pages, *information pages*, *Public interaction pages*, and *Private interaction pages*. *Information pages* provide information on different aspects of CDMT, such as supported *Stores*, an explanation of the *Decision Support* and so on. This pages can be viewed by any visitor and do not require

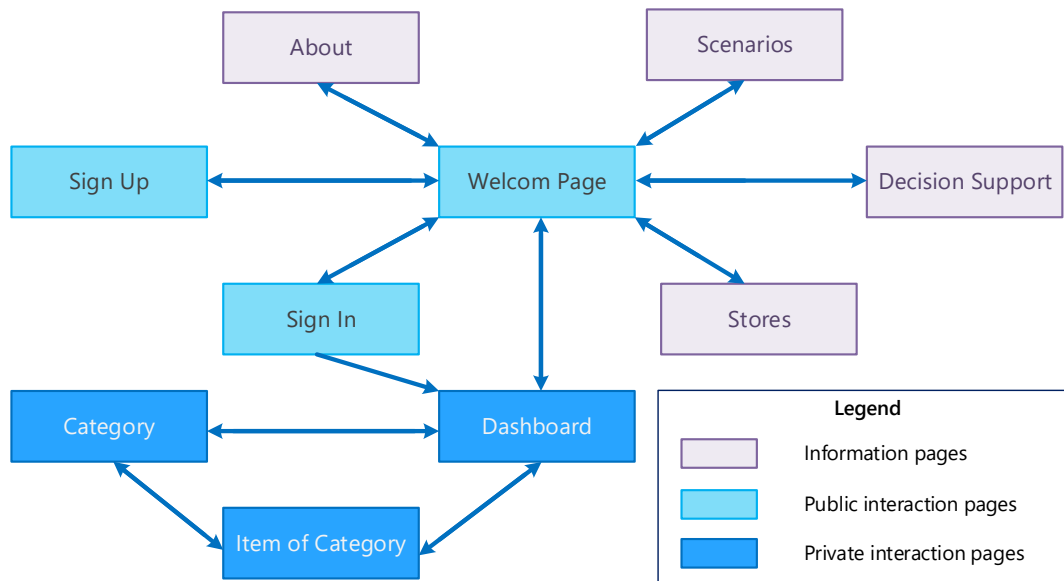


Figure 6.5.: Navigation Concept of CDMT

any credentials. The next category is *public interaction pages* which allow users to register themselves, or already registered users to enter the *private interaction pages*.

That pages enclose the functionality of CDMT. Also, it should be noted that the rectangles *Category* and *Item of Category* are a summary of different pages. Since these categories have the same generic concept and possibility of navigation we do not depict them explicitly for the sake of a clear illustration of the concept. Though, it is important to know that CDMT has three different categories, namely *Adapters*, *Projects*, and *Stores*. We will cover *Adapters* in the following subsections, but for more information on *Stores* and *Projects* we recommend to read [Bac12].

#### 6.4.2. Dashboard

The main page for registered users who are signed in is the *Dashboard*. In this section we explain the major elements and the their functionality of that page depicted in Figure 6.6. We marked the areas by rectangles. The upper one, marked with the number "1" in a circle, contains the *Navigation Bar*, the lower one contains the *Shorthand Tables*.

##### Navigation Bar

The Navigation Bar have all *private interaction pages* in common. In the list bellow we explain their functionalities beginning with the most left.

1. **Home Button:** From every page it navigates to Dashboard.

## 6.4. Extension of CDMT

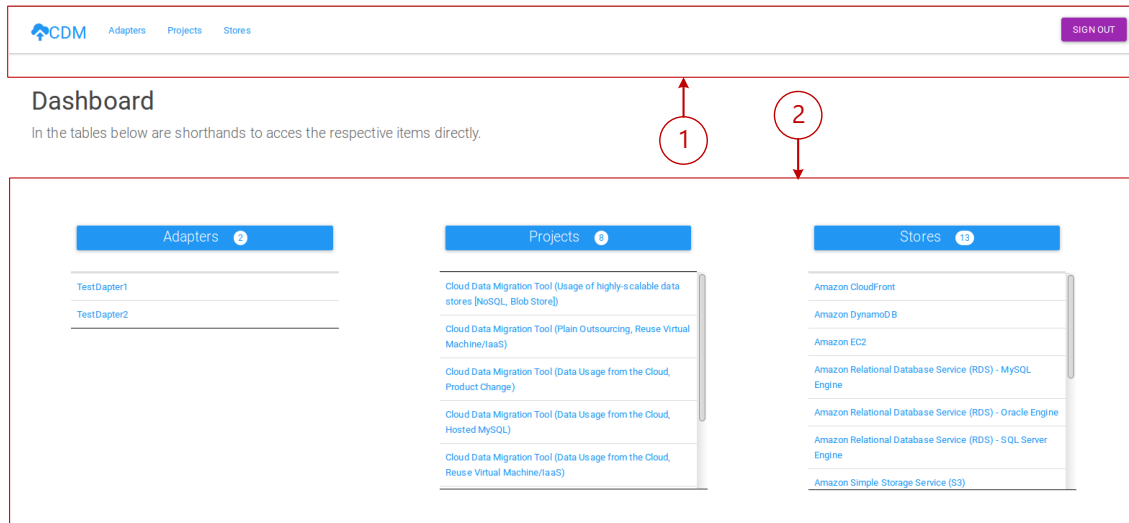


Figure 6.6.: Dashboard of CDMT

- 2. Adapters Page:** Navigates to the *Adapter Overview Page*, which delivers an detailed overview connectivity adapters and certain functions, that are available for the respective user.
- 3. Projects Page:** Opens the *Projects Overview Page* which provides a detailed information of projects of the specific user.
- 4. Stores Page:** Navigates to the *Stores Overview Page* that were created by users and are supported by CDMT.
- 5. Sign Out Button:** Signs out the user and shows the *Welcome Page*. Besides, it ends the user session.

### Shorthand Tables

The main benefit of the *Dashboard* are *Shorthand Tables*. They provide summarized overview of the items a specific user can access. By clicking on an item in the table the user will be redirected to its detailed page respectively. In the blue area above table content is the name of items' category with a count of entries in the respective table. The tables are listed below with the order from left to right.

1. Adapter Shorthand Table
2. Project Shorthand Table
3. Store Shorthand Table

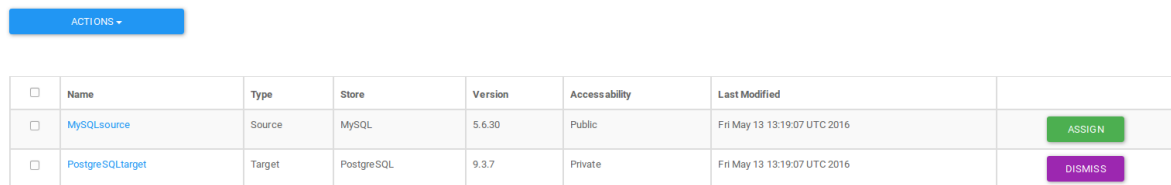
### 6.4.3. Adapter Overview Page

Adapter Overview Page is the Web-UI belonging to the new function of adding new connectivity adapters during runtime. Figure 6.7 depicts the default view for available adapters.

The blue button above the table on the left side opens a dropdown menu which provides actions to *Add Adapter* or to *Delete Selected*. The columns of the table are explained in the list below beginning with the most left.

#### Adapters

In the table below are the adapters you can use.



<input type="checkbox"/>	Name	Type	Store	Version	Availability	Last Modified	
<input type="checkbox"/>	MySQLsource	Source	MySQL	5.6.30	Public	Fri May 13 13:19:07 UTC 2016	ASSIGN
<input type="checkbox"/>	PostgreSQLtarget	Target	PostgreSQL	9.3.7	Private	Fri May 13 13:19:07 UTC 2016	DISMISS

Figure 6.7.: Adapters Overview

1. **Checkbox:** Allows to select adapters which should be deleted. If the "checkbox" in table's head is checked all entered adapters are selected.
2. **Name:** Contains the name of respective adapters. By clicking on the name the user is redirected to a detailed page of the selected adapter which shows all its entered properties.
3. **Type:** This column defines if an adapter is for connecting to a target, or to a source store for data migration.
4. **Store:** Entries in this column define the store type to which a specific adapter is able to connect.
5. **Version:** This column defines the version of the store which to for which a certain adapter is implemented.
6. **Availability:** Basically we distinguish between two availabilities, *Private* and *Public*. Private adapters can only be used and viewed by the user who added them. Other users do not see this adapters in their table. Public adapters are available for every user. By classifying these two availabilities we support *Security* to a certain extend (see Sect. 4.3.7).
7. **Last Modification:** Last modification shows the date on which the respective adapter was modified the last time.
8. **Assignment Buttons:** In this column a user is able to decide which of the available adapters he wants to use. By clicking on an *Assignment Button* which is highlighted by green colour and contains the value *Assign* the adapter will be added to CDMT's "classloader" and is considered as assigned by the user. The assignment state is saved persistently until it is changed by the user. If an adapter is not longer required it can be

dismissed by the user by clicking on the violet highlighted Assignment Button with the label *Dismiss*.

A further new feature is the *Delete Dialog*. It is the yellow rectangle with two buttons, which have the values *No* and *Yes*, and is depicted in Figure 6.8.

## Adapters

In the table below are the adapters you can use.

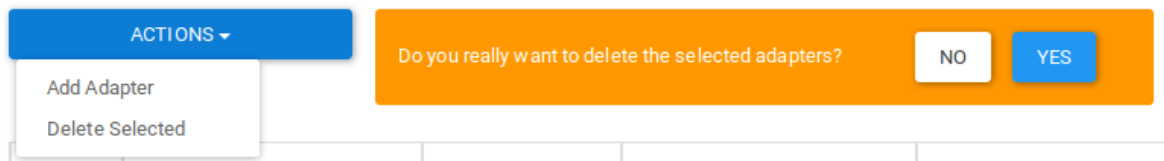


Figure 6.8.: Available Actions

The *Delete Dialogue* fades in if the user has selected one or more adapters and clicked on the *Delete Selected* action. By using this method we increase *Usability* (see Sect. 4.3.2) and prevent an unwanted behaviour. The motivation to use this approach was that the actions *Add Adapter* and *Delete Selected* are quite narrow. Hence, we wanted to prevent deletions by accident. If the user clicks on the button *No* nothing will happen but the fading out of the dialogue and the default state, as depicted in Figure 6.7, will be restored.

Just if the button *Yes* is clicked a request will be sent to the server, which initiates deletion of adapters properties from the DB and of the adapter's binaries from the hard disk drive. This action is irreversible. Selecting the action *Add Adapter* will redirect the *New Adapter Page* (see Sect. 6.4.4).

Another feature is user feedback when assigning or dismissing an adapter. If the desired action is executed successfully an information panel on at the location the *Delete Dialogue* is displayed in Figure 6.8 an information fades in. After three seconds it fades out automatically.

### 6.4.4. New Adapter Page

An essential Web-UI component to use Adapters is the *New Adapter Page*. Figure 6.9 depicts this page which consists of a form. At the top left corner is a *Back Arrow Symbol* which provides the possibility to cancel the operation and to return to the *Adapter Overview Page*. We give an explanation of the fields in the form in the list below, whereas the order is from the top to the bottom field, see Figure 6.9. Generally we designed it lean by giving a field's purpose within itself to avoid an overloaded design. This entry disappears as soon as the user starts entering another value. The content of *New Adapter Page* is used, if detailed information on an available adapter shall be shown after it is added as well.

The screenshot shows a web form titled "New Adapter". At the top left is a back arrow icon. The form fields are as follows:

- Adapter Name:** A text input field with a key icon on the left.
- Database Type:** A text input field with a list icon on the left.
- Store Version:** A text input field with a key icon on the left.
- Describe the Adapter here...:** A text area with a pencil icon on the left.
- Select Adapter Type:** A dropdown menu with a square icon on the left and a downward arrow on the right.
- Select Accessibility:** A dropdown menu with a lock icon on the left and a downward arrow on the right.
- File Upload:** A large dashed rectangular box containing the text "Drag & drop files here ...". A small "X" icon is in the top right corner of the box. Below the box is a blue button with a folder icon and the text "BROWSE ...".
- ADD ADAPTER:** A blue button at the bottom left of the form.

**Figure 6.9.:** Page for Adding a new Adapter

1. **Adapter Name:** The user has to enter a name for the adapter as text.
2. **Database Type:** The user should enter a database type. This information is useful for other users, since it defines the database type to which the adapter is able to connect.
3. **Store Version:** The entry in this field allows the user to enter the version of the store for which the adapter is implemented.
4. **Description Area:** Provides the possibility to give further and detailed information about the adapter. This area is restricted to alphanumerical entries only.
5. **Select Adapter Type:** By clicking on this field a drop-down-menu appears below which has two options *Source* and *Target*. This information selecting one of these options

is mandatory, since for the data model it is necessary to know in order to load any adapter to the "classloader".

6. **Select Accessibility:** By clicking on this field a drop-down-menu appears which provides the options *Private* and *Public*. It is obligatory to select one of these options, since it decides if the adapter can be used by every registered user or by the owner only.
7. **File Upload Area:** To add an adapter to CDMT a binary file must be uploaded to the server first. This area provides two different possibilities to upload an adapter. The user can either just drag the file and drop it to the area containing the information "*Drag & drop files here ...*", or to click on the *Browse Button*, highlighted by a folder symbol. Clicking on this button opens a file Browser which is used to navigate to the directory of the desired file and selecting it. In both cases the *File Upload Area* shows the selected file and additional buttons as depicted in Figure 6.10. The buttons are *Remove* and *Upload*, their functionality is obvious by their names. Clicking on *Upload* the user gets feedback on the operation's outcome, if it was successful the file is stored in the directory for adapters on the sever and the *Add Adapter Button* will be enabled. In other cases an error description will appear. The upload functionality is restricted to one file per adapter and to JAR-files only.
8. **Add Adapter Button:** The *Add Adapter Button* is disabled as default, and will be enabled if a file is successfully uploaded. When it is enabled the entered values in the fields above are validated. If a mandatory required value is not given the respective field is highlighted and an information to fill that field. However, if the form is filled correctly the all entries for the adapter are stored in the db, the user is redirected to the *Adapter Overview Page*, the added adapter is shown in the *Adapter Table*, and can be used.



Figure 6.10.: File Upload Area in Detail

#### 6.4.5. Approval of Implementation and Integration

To integrate the extensions into CDMT we use the top-down approach (see Sect. 2.4.2). We outline the tasks we performed and the order in which the components are added in this subsection. The validation of the entire system is the subject of Chapter 7. Thus, at this point we will cover the approval of the components. The Java components are tested with each build of CDMT in an isolated manner by Unit Tests. However, it enabled the possibility to test CDMT semi-automatically.

##### Web-UI Components

Since our *Web-UI Components* are just to visualise the content and do not have any processing logic we had to prove their behaviour and links between each other. Additionally we used some dummy data to ensure a correct depiction of the information and design. We added these components first since we stick to the integration process, to have control in each state and be able to identify causes for misbehaviour more precisely.

##### Servlets

*Servlets* are responsible for the communication between client and server. They do not access any resources wherefore they were integrated after the *Web-UI Components*. By using empty messages we are able to test if the connection is established correctly. Afterwards we used static values to ensure correct message format.

##### Classes for Persistent Java Objects

The lowest components in our architecture of the top-down perspective are *Classes for Persistent Java Objects* which are responsible for data access and manipulation. Hence, failures at this place are quite critical. We minimized the risks since we isolate CDMT execution environment and can reproduce a clean state of resource data at any time (see Sect. 6.2.4) These are components we added last during the integration.

The overall system test is part of the validation (see Chap. 7). This final test is important to exclude side effects on the CDMT's components that were already integrated.

### 6.5. Exemplary Adapters

In this section, we will outline the implemented adapters. We provide a MySQL source adapter that is introduced in Subsection 6.5.1 to obtain the data. In Subsection 6.5.2, we briefly present the PostgreSQL target adapter which is responsible for writing data to the target store. For both adapters we use JDBC to reduce dependencies for the installation and configuration of the runtime environment and its installations. We do not explain the functionality or features of JDBC, yet we suggest to read [Ora] for more information on this topic. The JDBC driver and its version, that we use, are declared as a dependency in the POM file



## 6.5. Exemplary Adapters

---

of every adapter. Moreover, for every adapter we create a separate project. Furthermore, we take the already existing adapters of CDMT, which are part of the source code, as an example to provide adapters which are as consistent as possible to CDMT's design and implementation. Hence, we facilitate the usage of the provided interfaces for both types of adapters (see Appendix A).

### 6.5.1. MySQL Source Adapter

We describe shortly what is important to consider in an implementation of a source adapter and what differs from the previous implementations. Since we use JDBC to connect to a source store, we need to create a *connectivity object* at the beginning of a session. Since such connection objects can be used for various SQL store types, it is important to specify "jdbc:mysql:" at the beginning of the connection string to load the right driver. Afterwards, we open a connection with properties, which we obtain from the user. Transactions between CDMT and the respective store are handled by this object. Additionally, we have to specify and create a directory which we use to store the content of a source store. After we transfer the data to that directory we consider all transactions finished. It is essential to close every connection for the sake of security, regardless of whether the data was transferred successfully or not.

### 6.5.2. PostgreSQL Target Adapter

In addition to the connection handling we have described in section 6.5.1, we introduce characteristics which differ from the source adapter. A major one is to implement a method to read the data transferred from the source store. Another is to convert the data into a convenient format for the target store. After CDMT terminates migration, the directory for data to export to and its content must be removed in any case.



---

## 7. Validation

---

Every change to a system might have unforeseeable side effects and misbehaviour. Hence, we always have to prove the concepts which we used in this work and the implementation. In this chapter, we present the tools, data, and method we used to validate our implementation. In Section 7.1, we give an overview of the tools we use for our validation. To run our test, we generate data which we outline in Section 7.2. We explain in Section 7.3 the use cases which we used for validation. Finally, we present the validation and the insights we gained from it in Section 7.4.

### 7.1. Tool and Services

In this section, we give an overview of additional tools that are required for our validation. Furthermore, we have to use services that provide a Cloud Computing environment. We also explain for which purposes we use each tool or service.

#### **Amazon RDS**

Amazon RDS provides the possibility to scale a relational DB in a Cloud environment. It supports various RDBMS per default. We leverage this property to create an instance of PostgreSQL which we use as target store for our validation.

#### **Apache JMeter<sup>1</sup>** (Version: 2.13)

Apache JMeter has the ability to test different kinds of resources. Since it can be configured to send requests with predefined queries to a specified DB, we use it to prove the consistency between source and target data. Furthermore, it enables to run such tests very frequently. Thus, we are able to rerun the tests in case of changes in a short time once it is set up.

#### **pgAdmin<sup>2</sup>** (Version: 1.20.0 Beta 2)

pgAdmin is an open source tool with a GUI which provides functionalities to manage and develop PostgreSQL DBs. We use it to examine the target store if our automated requests do not deliver the desired results. Additionally, it is useful for troubleshooting if we observe misbehaviours, such as connectivity problems or unavailability of the target store.

---

<sup>1</sup><http://jmeter.apache.org>

<sup>2</sup><http://www.pgadmin.org>

**TPC-H<sup>3</sup>** (Version: 2.15)

*"TPC-H is decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications."* [Tra14]. We use it to generate our test data which is stored in the test DB. Since we do not use further functionalities, we will not give a detailed introduction, but refer to [Tra14] for further information.

**7.2. Test Data**

In this section, we present the data we use for our validation. Figure 7.1 depicts the crowfoot diagram of the MySQL DB with an overall size of one gigabyte. We generate the data with *TPC-H* and chose this schema on purpose, because it has a certain complexity to approximate a real-live scenario. We use as similar schema and data as it is used in [Lam13]. Furthermore, it was important to examine how CDMT handles different data types and relations of a certain DB. Additionally, we are able to review information on restrictions in a DB, such as columns for PK, FK, or fields which must not be empty.

---

<sup>3</sup><http://www.tpc.org/tpch>

## 7.2. Test Data

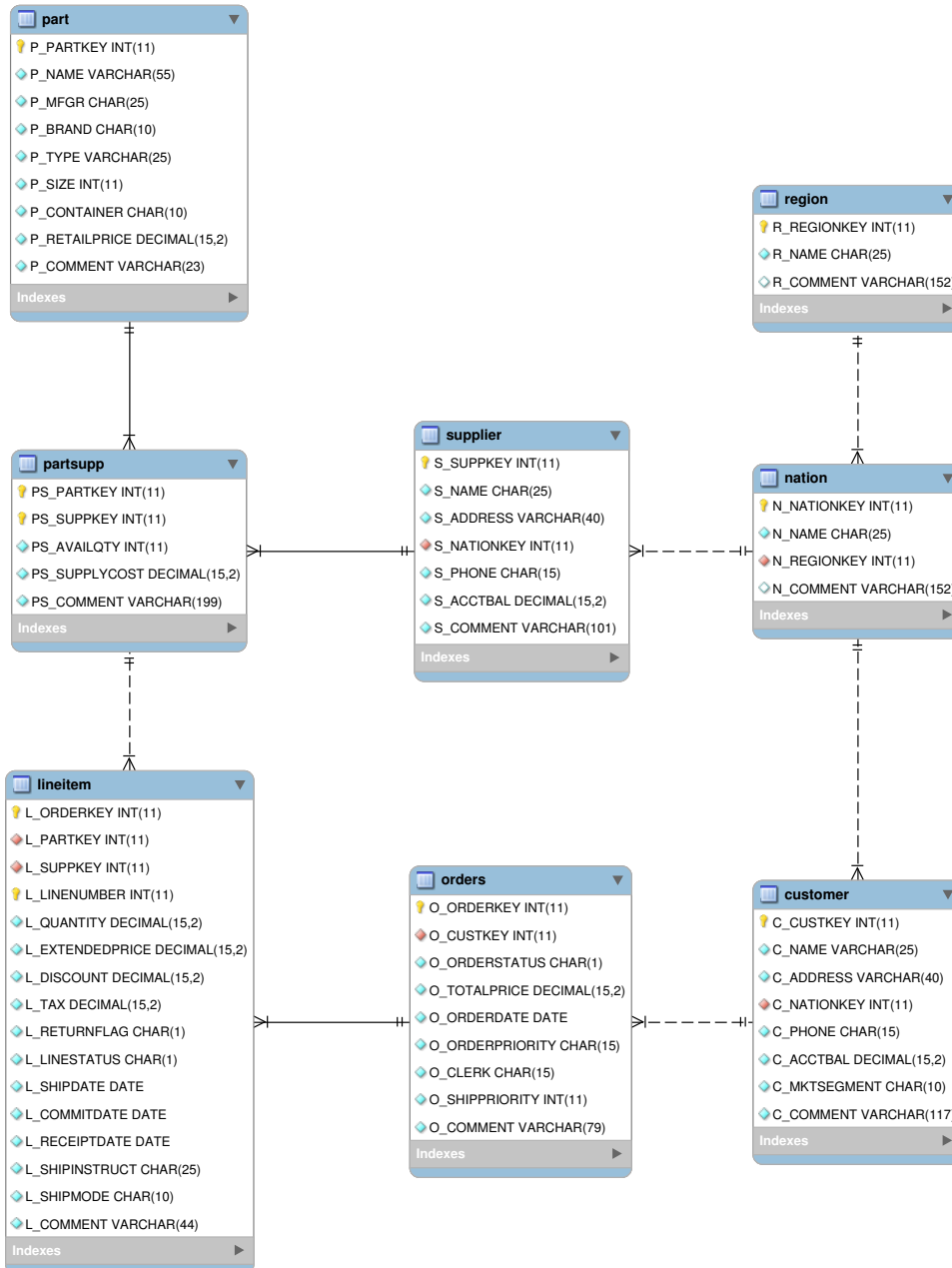


Figure 7.1.: Crowfoot Diagram of the Test DB, Generated With MySQL Workbench

### 7.3. Test Cases

We have to define which aspects of CDMT we want to validate and which mechanisms are suitable. In this section, we introduce the test cases which we used for our validation. Generally, we use two kinds of test cases. In Section 7.3.1, we explain the connection between use cases we defined in Section 4.2 and how we leverage that for validation. Afterwards we describe how we use previous results (see Sect. 3.1) to extend the validation in Section 7.3.2.

#### 7.3.1. Use Case Based

Use cases specify expected behaviour of an entire system or components. Hence, they can be used to derive test cases directly. Furthermore, we utilise them to validate the functions which we implemented and their behaviour in the context of the overall system. Additionally, they complete the process of integration (see Sect. 6.4.5). We consider the specified regular cases as the expected results for successfully accomplished test cases.

In the event of unexpected behaviour, or the specified exceptional cases we use the following method to correct it:

1. Identify the cause of misbehaviour.
2. Eliminate what is responsible for the error(s).
3. Execute all operations of the test case again.

We iterate this method until errors and failures that appear during the execution of the respective test cases are resolved.

#### 7.3.2. Based on Previous Results

We use the validation from [Lam13] as well as the evaluations from [Guo13], and [Nac13] to define a configuration for this test case. Moreover, we prove the functionality of our test adapters (see Sect. 6.5). We define a detailed set-up for the test case migration in Table 7.1.

<b>Name:</b>	Migrate Data (manual interaction required)
<b>Goal:</b>	Migrate test data (see Sect. 7.2) without loss.
<b>Source Adapter:</b>	MySQL (JDBC version: 5.1.38.JDBC4) (see Sect. 6.5.1)
<b>Target Adapter:</b>	PostgreSQL (JDBC version: 9.3.-1100-JDBC41) (see Sect. 6.5.2)
<b>Source Store:</b>	Local MySQL (version: 5.6.30)
<b>Target Store:</b>	Hosted by Amazon RDS PostgreSQL (version: 9.3)

<b>Migration Scenario:</b>	Plain Outsourcing
<b>Required Input:</b>	<ol style="list-style-type: none"><li>1. Properties to connect to the source store (Host, Port, DB-name, User-name, Password).</li><li>2. Properties to connect to the target store (Host, Port, DB-name, User-name, Password).</li></ol>
<b>Pre-Conditions:</b>	Accomplish use case based tests (see Sect. 7.3.1) successfully.
<b>Post-Conditions:</b>	<ol style="list-style-type: none"><li>1. The data is available in the source and the target store.</li><li>2. The export directory is removed.</li></ol>
<b>Successfull Result:</b>	The content of the source DB and its data model is migrated successfully.
<b>Failed Results:</b>	<ol style="list-style-type: none"><li>1. The data is migrated without its model.</li><li>2. The data is not migrated, but the model is.</li><li>3. No migration at all.</li><li>4. Connection to the source store cannot be established.</li><li>5. Connection to the target store cannot be established.</li></ol>

**Table 7.1.:** Test Case Migration

We chose *Plain Outsourcing* (see Table 2.2) as scenario since it is validated in [Lam13]. Although, we use RDBMS of different vendors to extend the validated configurations. Furthermore, it is essential that before this test case is examined all use case based tests complete successfully since it ensures that the adapters, which we implemented as plug-ins can be used by CDMT.

Neither do we describe the execution of this in detail, nor do we describe our selected options defined by the methodology (see Sect. 2.3.2) since it does not have impacts on the outcome of the validation. For detailed information on this topics we recommend [Bac12] and [Lam13]. Moreover, successfully completing this test ends the validation and integration.

## 7.4. Validation Result

In this section, we present the results of the validation. In Section 7.4.1, we outline the outcome and problems we had during performing tests based on use cases. Finally, we explain the outcome of the tests based on previous results in Section 7.4.2.

### 7.4.1. Outcome of Use Case Based Tests

The first test run delivered a server side error. We were able to localise it in the servlets. The cause was an inconsistent message format between client and server. We corrected the message processing and moved on. The second run was unsuccessful as well. Due to a syntax

error in the queries that are responsible for DB entries we got an `SQL_Exception`. We adapted the queries and rebuilt the entire system. Finally the third run was successful and all test cases were accomplished as expected.

#### **7.4.2. Outcome of Tests Based on Previous Results**

The execution of this phase of the validation was critical. Although, we were able to load our adapters and connect to the stores, we could not migrate the test DB at all. We had to inspect the source code of our adapters and components which interact with them in detail.

After cumbersome debugging, we are led to the conclusion that the implementation of our connectivity adapters was not sophisticated enough to handle the data migration. We found out that it was not possible to write fully operational adapters as part of this work since the default functionality of JDBC-Drivers does not cover all possibilities of the previously used proprietary products. On the one hand, we were able to obtain data from the source store and store it in files in simple formats. On the other hand, JDBC-Drivers have no build-in functions to obtain the schema nor the relation definition between the different entities. All this functionality has to be implemented additionally. Hence, we have a negative result for our attempt to exchange the source and target adapters of CDMT entirely by JDBC based solutions.



---

## 8. Conclusion and Future Work

---

In this final chapter, we summarize our efforts and the conclusion we gained during this work. We outline recommendations for further improvements of CDMT and ideas for future work.

After we introduced the concepts, we conducted an analysis on how to include them in the prototype and specified its requirements. We reused previously implemented prototypes to reduce redundant labour. Additionally, we specified new functionalities to enable not only a broader use of CDMT prototype but also to prepare it for extensibility so it can be extended more easily. After our initial analysis we redesigned and extended the architecture of CDMT. We decided which components could be reused, which would have to be exchanged for integrating them into the final prototype and enable extensibility. After the integration phase, we gained the insights on obsolete technologies that we used, as well as their impact on the development process, from a software-engineering perspective. During the implementation encountered problems which could not be resolved by design. We provided an efficient and defined build cycle which increased productivity and transparency for this phase. Furthermore, we prepared our prototype to be used in several different environments.

Afterwards, we provided a modern design for a better user experience and appropriate appearance. We were able to implement a plug-in mechanism for a high degree of customisation and extensibility. Moreover, this property enables the support for future types of stores or new implementations of existing stores.

We were able to identify possibilities to increase the potential of CDMT left for future work.

For future work, we suggest to provide extended interfaces for connectivity adapters. We have to mention that a more complex implementation of such adapters is still outstanding. Hence, it is recommended to provide methods which implement, or even outclass, functionalities of proprietary solutions. An other interesting investigation in this context is to distribute the data of a source store on multiple stores of a different kind.

Although, this attempt requires further research on topics like information densities of certain tables in a schema of a RDBMS, and which impact it has if the data is migrated to another store type. Yet another considerable task is to identify equivalences in models and implement different store types to transformation patterns for DB-designs during a migration to another data model.



---

## Appendix A.

### Interfaces

---

#### A.1. Source Adapter Interface

---

```
1 package com.clouddatamigration.migration.model;
2
3 public interface SourceSystem {
4
5     String getId();
6
7     ArrayList<String> getConnectionParameters();
8
9     boolean connect(HashMap<String, String> connectionProperties,
10 ServletOutputStream out);
11
12     String getInstructions();
13
14     /**
15      * Returns SQL commands to recreate the tables and their data in a
16      * relational database
17      *
18      * @param out
19      * @return
20      */
21     String getSqlCommands(ServletOutputStream out);
22
23     /**
24      * Returns a map of tables with their exported data as CSV
25      *
26      * @param out
27      * @return key = table name, value = CSV with headers in first line, values
28      *         in following lines
29      */
30     HashMap<String, String> getTablesAsCSV(ServletOutputStream out);
31 }
```

---

**Listing A.1:** Source System Interface, Part of [Bac12]

---

## A.2. Target Adapter Interface

---

```
1 package com.clouddatamigration.migration.model;
2
3 public interface TargetSystem {
4
5     /**
6      * Returns a unique ID that also describes the adapter, e.g.
7      * AWS_RDS_MYSQL_TARGET
8      *
9      * @return
10     */
11     String getId();
12
13     /**
14      * Returns true if SQL commands are supported and false if a CSV data import
15      * is required
16      *
17      * @return
18     */
19     boolean supportsSql();
20
21     /**
22      * Returns a list of required or optional connection parameters, the meaning
23      * of each parameter and whether it is required or not is described in the
24      * instructions.
25      *
26      * @return
27     */
28     ArrayList<String> getConnectionParameters();
29
30     /**
31      * Tries to connect to the data store and makes sure that not only the
32      * credentials (connection properties) are valid, but also that some kind of
33      * action can be called that does not change anything.
34      *
35      * @param connectionProperties
36      * @param out
37      * @return true if successful, false if not
38     */
39     boolean connect(HashMap<String, String> connectionProperties,
40                     ServletOutputStream out);
41
```

## A.2. Target Adapter Interface

---

```
42 /**
43  * Migration using SQL commands, tries to preserve constraints
44  *
45  * @param sqlCommands
46  * @param resp
47  *      {@link HttpServletResponse} with the possibility to write to
48  *      the output stream, send files or redirect the user
49  * @return true if successful, false if not
50  */
51 boolean migrate(String sqlCommands, HttpServletResponse resp);
52
53 /**
54  * Returns printable (html) instructions what the connection parameters
55  * mean, what the import adapter does and what it is capable of.
56  *
57  * @return
58  */
59 String getInstructions();
60
61 /**
62  * Migration using CSV format, ignores constraints
63  *
64  * @param csvTables
65  * @param resp
66  *      {@link HttpServletResponse} with the possibility to write to
67  *      the output stream, send files or redirect the user
68  * @return true if successful, false if not
69  */
70 boolean migrate(HashMap<String, String> csvTables, HttpServletResponse resp)
71     ;
72 String getTargetStoreType();
73 String getTargetId();
74 }
```

---

**Listing A.2:** Target System Interface, Part of [Bac12]



---

## Bibliography

---

- [ABLS13] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch. How to Adapt Applications for the Cloud Environment. *Computing*, 95:493–535, 2013.
- [apc] apache.org. Apache Maven Project. <https://maven.apache.org/index.html>.
- [Bac12] T. Bachmann. Entwicklung einer Methodik für die Migration der Datenbankschicht in die Cloud. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, September 2012. No. 3360, [http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL\\_view.pl?id=DIP-3360&mod=0&engl=0&inst=IAAS](http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3360&mod=0&engl=0&inst=IAAS).
- [Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, June 1970.
- [Cor] O. Corporation. Lesson: Creating and Using Extensions. <https://docs.oracle.com/javase/tutorial/ext/basics/index.html>.
- [doc] docker.com. Docker Engine. <http://www.docker.com/products/docker-engine>.
- [Far13] S. Faroult. *SQL Success*. Roughsea Limited, 2013.
- [FLR<sup>+</sup>14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. *Cloud Computing Patterns*. Springer Wien, Januar 2014.
- [Gro09] T. Group. *PostgreSQL 8.4 Official Documentation - Volume II. Server Administration*. Fultus Corporation, 2009.
- [Guo13] X. Guo. Evaluation of a Methodology for Migration of the Database Layer to the Cloud based on an eScience Case Study. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, December 2013. No. 3512, [http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL\\_view.pl?id=DIP-3512&engl=1](http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3512&engl=1).
- [jav] java2s.com. Maven Tutorial. [http://www.java2s.com/Tutorials/Java/Maven\\_Tutorial](http://www.java2s.com/Tutorials/Java/Maven_Tutorial).
- [Lam13] R. Lamllari. Extending a Methodology for Migration of the Database Layer to the Cloud Considering Relational Database Schema Migration to NoSQL. Master’s thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, June 2013. No. 3460, [http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL\\_view.pl?id=MSTR-3460&mod=0&engl=0&inst=IAAS](http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=MSTR-3460&mod=0&engl=0&inst=IAAS).

- 
- [Lig09] P. Liggesmeyer. *Software-Qualität : Testen, Analysieren und Verifizieren von Software*. Heidelberg : Spektrum, Akad. Verl., 2009.
- [LL10] J. Ludewig and H. Lichter. *Software Engineering : Grundlagen, Menschen, Prozesse, Techniken*. Heidelberg : dpunkt-Verl., 2010.
- [MG11] P. Mell and T. Grance. The NIST Definition of Cloud Computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.
- [MMB09] T. Malone, I. Menken, and G. Blokdijk. *Itil V3 Foundation Complete Certification Kit: Study Guide Book and Online Course*. Art of service. 2009.
- [Mor12] J. Morris. *Practical Data Migration - Second edition*. British Informatics Society Limited, 2012.
- [Nac13] N. Nachev. Evaluierung einer Methodik für die Migration der Datenbankschicht in die Cloud basierend auf einer Fallstudie aus der Industrie. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, August 2013. No. 3469, [http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL\\_view.pl?id=DIP-3469&mod=0&engl=0&inst=IAAS](http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3469&mod=0&engl=0&inst=IAAS).
- [O'N14] P. O'Neil. *Database: Principles Programming Performance*. Elsevier Science, 2014.
- [Ora] Oracle. Java JDBC API. <http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc>.
- [SAB<sup>+</sup>13] S. Strauch, V. Andrikopoulos, T. Bachmann, D. Karastoyanova, S. Passow, and K. Vukojevic-Haupt. Decision Support for the Migration of the Application Database Layer to the Cloud. In *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'13)*, pages 639–646. IEEE Computer Society Press, December 2013.
- [SAGSL13] S. Strauch, V. Andrikopoulos, S. Gómez Sáez, and F. Leymann. ESB<sup>MT</sup>: A Multi-tenant Aware Enterprise Service Bus. *International Journal of Next-Generation Computing*, 4(3):230–249, 2013.
- [SAK<sup>+</sup>14] S. Strauch, V. Andrikopoulos, D. Karastoyanova, F. Leymann, N. Nachev, and A. Staebler. Migrating Enterprise Applications to the Cloud: Methodology and Evaluation. *International Journal of Big Data Intelligence*, pages 1–20, 2014.
- [SALM12] S. Strauch, V. Andrikopoulos, F. Leymann, and D. Muhler. ESB<sup>MT</sup>: Enabling Multi-Tenancy in Enterprise Service Buses. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'12)*, pages 456–463. IEEE Computer Society Press, December 2012.
- [SF12] P. J. Sadalage and M. Fowler. *NoSQL Distilled: a Brief Guide to the Emerging World of Polyglot Persistence*. Pearson Education, 2012.



## Bibliography

---

- [Tra14] Transaction Processing Performance Council (TPC). *TPC BENCHMARK<sup>TM</sup>H - Standard Specification*, 11 2014. Rev. 2.17.1.
- [Vai13] G. Vaish. *Getting Started with NoSQL*. Packt Publishing, 2013.
- [VMZ<sup>+</sup>10] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective. In *Proceedings of the 48th Annual Southeast Regional Conference, ACM SE '10*, pages 42:1–42:6, 2010.
- [w3s] w3schools.com. AJAX Tutorial. [http://www.w3schools.com/ajax/ajax\\_intro.asp](http://www.w3schools.com/ajax/ajax_intro.asp).
- [Yim] T. Yim. Java Class Loading and Distributed Data Processing Frameworks. <http://blog.cask.co/2015/08/java-class-loading-and-distributed-data-processing-frameworks/>.

All links were last followed on May 18, 2016.

## **Acknowledgement**

I am heartily thankful to my supervisor Steve Strauch from the University of Stuttgart for his encouragement, guidance and support in all the phases of this diploma thesis. I am also grateful to Dr. Vasilios Andrikopoulos for his advices and useful tips. Special thanks to my family and friends for their moral support.

Andreas Rempel

## Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any sources and references other than those listed. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, 20th May 2016

\_\_\_\_\_  
(Andreas Rempel)