TU Clausthal

Stefan T. Ruehl

Mixed-Tenancy Systems

A hybrid Approach between Single and Multi-Tenancy

SSE-Dissertation 9

Software
Systems
Engineering

Department of Informatics
Chair of Prof. Dr. Andreas Rausch

Mixed-Tenancy Systems

A hybrid Approach between Single and Multi-Tenancy

D o c t o r a l T h e s i s
(D i s s e r t a t i o n)

to be awarded the degree of
Doctor of Engineering
(Dr.-Ing.)

submitted by
Stefan T. Ruehl
from Darmstadt

approved by the Department of Informatics,
Clausthal University of Technology

2014

Dissertation Clausthal, SSE-Dissertation 9, 2014

Date of oral examination: June 16, 2014

Abstract

Multi-Tenancy is an architectural paradigm that is supposed to allow operators to exploit economies of scale. This is due to the fact that a single instance of Multi-Tenancy application serves multiple customers at the same time. Thus, operators may utilize resources and facilitate application operations more efficiently. On the other hand, however, a major drawback of Multi-Tenancy is the customers' hesitation of sharing infrastructure, application code, or data with other tenants. According to recent studies, this is due to the fact that one of the major threats of Multi-Tenancy is information disclosure due to a system malfunction, system error, or aggressive actions by individual users. So far, the only approach in research to counteract on this hesitation has been to develop new techniques to enforce the isolation between tenants using the same instance.

This work tackles this challenge by proposing a novel approach that is referred to as Mixed-Tenancy. It allows customers to express their deployment constraints about if or even with whom they want to share the application. To be more precise, the approach enables the customer to make that choice not just for the entire application but specifically for individual application components and their underlying infrastructure stack. Based on these constraints a deployment is computed that uses infrastructure as efficiently as possible by being in compliance with all constraints. Such a deployment is referred to as valid and optimal. Thus, Mixed-Tenancy is an approach that allows operators to exploit economies of scale by still keeping customers' hesitations concerning the privacy threats of Multi-Tenancy in mind.

This work contributes to the creation of Mixed-Tenancy systems by introducing a generic model that allows capturing customers' deployment constraints. Thereby, the model allows customers to express complex constraints (e.g. "sharing shall only be permitted with companies from Europe but not with competitors") while still allowing the operator to keep its customer base secret. In addition the problem of computing a valid and optimal deployment is formally defined and analyzed. Furthermore, it is proven to be NP-hard and two intuitive heuristics are introduced and compared.

Finally, this work evaluates the applicability of Mixed-tenancy by investigating a case study in the area of cloud computing. This is done by introducing Mixed-Tenancy to an existing cloud application, called OpenERP, currently used in industry. It thereby demonstrates that the Mixed-Tenancy approach may indeed be successfully applied to real-world systems.

Für Carina (Bald-)Ruehl

Danksagung (Acknowledgments)

Nothing of me is original. I am the combined effort of everybody I've ever known.

Chuck Palahniuk

Diese Arbeit ist nicht das Werk einer einzelnen Person, sondern wäre nicht möglich gewesen, ohne die Unterstützung vieler. Hiermit möchte ich mich bei einigen noch einmal besonders bedanken.

Ich bedanke mich bei Prof. Dr. Urs Andelfinger und Prof. Dr. Andreas Rausch, meinen Doktorvätern, die mir die Möglichkeit gegeben haben, eine berufsbegleitende Promotion durchzuführen und mich dabei unterstützt haben. Mir wurde durch meine Doktorväter eine exzellente Umgebung für die Entwicklung dieser Arbeit geschaffen, in der es möglich war, im engen Dialog neue Ideen offen zu diskutieren. Ich habe alle unserer vielen, teilweise kritischen, Diskussionen als extrem fruchtbar und bereichernd empfunden.

Des Weiteren möchte ich mich herzlichst bei Prof. Dr. Steffen Lange und Prof. Dr. Marian Margraf von der Hochschule Darmstadt bedanken. Ohne sie wären die formalisierten Darstellungen und Beweise in dieser Arbeit in der vorliegenden Form nicht möglich gewesen. Bei Steffen Lange möchte ich mich besonders für seine Geduld bei unseren ersten Terminen bedanken. Auch wenn mir die Ästhetik von mathematischen Formalisierungen wahrscheinlich für den Rest meines Lebens in seiner Gänze verschlossen bleiben wird, hat er mir doch einen sehr guten Einblick gegeben.

Des Weiteren geht mein Dank an Prof. Dr. Holger Wache von der Fachhochschule Nordwestschweiz für seine Unterstützung bei der Realisierung des Description Models mit Hilfe von Technologien aus dem Gebiet des Semantic Webs. Es hat mich sehr gefreut, mit Dir daran zu arbeiten.

Diese Arbeit ist berufsbegleitend neben meiner Tätigkeit bei der T-Systems entstanden. Sie wäre sicherlich nicht möglich gewesen, hätte ich nicht auch durch meinen Arbeitgeber ideale Bedingungen vorgefunden. Besonders danken möchte ich hier meinem Vorgesetzten Dr. Stephan Verclas, Leiter des Innovation Centers der T-Systems. Er hat diese Bedingungen für mich maßgeblich geschaffen. Mein Dank geht aber auch an meine Kollegen Sascha Steiner, Jens Paprotny und alle meine anderen Kollegen aus dem Innovation Center dafür, dass sie mir vor allem gegen Ende der Promotionszeit genug Freiraum geschaffen haben, um die Arbeit meinen Vorstellungen entsprechend abzuschließen. Zudem geht mein Dank auch an Matthias Reinhard, Malte Rupprecht, Björn Morr und Candide Orou-Yorouba, die es geschafft haben, diese Arbeit durch die Arbeit an ihren Abschlussarbeiten und den damit verbundenen Diskussionen zu unterstützen und zu bereichern. Kapitel 6 und Anhang B habe ich maßgeblich mit Ihnen zusammen entwickelt.

Für die vielen inhaltlichen Diskussionen sowie die intensiven Reviews gegen Ende geht mein Dank an Dr. Marcus Zinn und Michael Roth.

Des Weiteren danke ich meinen Eltern, Antje und Bernd, sowie meinen Großeltern, Karla und Jochen, die mich während meiner gesamten schulischen und akademischen Ausbildung immer unterstützt und gefördert haben.

Letztendlich geht mein Dank an Carina Voigt, der ich diese Arbeit widme. Carina hat mir in allen Phasen dieses Projektes stets zur Seite gestanden, hat mich unterstützt und mir den Rücken frei gehalten. Sie hat stets meine Launen ertragen und mir geholfen, auch nach Rückschlägen doch nicht das Ziel aus den Augen zu verlieren. Ohne sie wäre die Arbeit nicht möglich gewesen.

Vielen Dank!

Stefan Tobias Ruehl
Erzhausen, den 18.04.2014

Contents

Abstract **v**

Contents **xi**

List of Figures **xv**

List of Tables **xvii**

List of Definitions **xix**

1. Introduction **1**
- 1.1. Motivation 1
- 1.2. Goals and Contributions of this Work 2
- 1.3. Structure and Content 3
- 1.4. Research Methodology 4

2. Towards Mixed-Tenancy - A Problem Analysis **9**
- 2.1. Fundamental Concepts and Approaches I 10
  - 2.1.1. Relevant Stakeholders 10
  - 2.1.2. Component-Based Software 11
  - 2.1.3. Single-Tenancy and Multi-Tenancy 12
- 2.2. Scope of this Work 14
  - 2.2.1. Introduction of Mixed-Tenancy 15
  - 2.2.2. Challenges of Mixed-Tenancy 17
  - 2.2.3. Research Questions of this Work 19
  - 2.2.4. Limitations of Scope 20
  - 2.2.5. Deployment Configuration Generator 22
- 2.3. Application of Mixed-Tenancy in Cloud Computing 23
  - 2.3.1. Introduction to Cloud Computing 23
  - 2.3.2. Application of Mixed-Tenancy in Cloud Computing 25
- 2.4. Summary 26

3. State of the Art **29**
- 3.1. Research related to Multi-Tenancy 29
  - 3.1.1. Realization of Multi-Tenancy 30
  - 3.1.2. Variability of Multi-Tenancy Applications 33

3.2. Related Approaches tackling Security Issue 36
3.2.1. Introduction of the Underlying Mapping Study 36
3.2.2. Related Work tackling Data Breach 37
3.3. Summary 39

4. Capturing Customers' Deployment Constraints 41
4.1. Fundamental Concepts and Approaches II 43
4.1.1. First-order Logic 44
4.1.2. Introduction of Graph Theory 44
4.2. Conceptual Analysis based on Requirements 46
4.2.1. Levels of Deployment 46
4.2.2. Deployment Models 49
4.2.3. Groups 51
4.2.4. Dimensions 51
4.2.5. Virtual Tenants 52
4.3. Process of Deployment Constraint definition 53
4.3.1. Customizing 53
4.3.2. Tenant Grouping 55
4.3.3. Constraint Definition 55
4.3.4. Deployment Information Extraction 56
4.4. Generic Mixed-Tenancy Description Model 58
4.4.1. Introduction of the Fundamental Idea 58
4.4.2. Foundations of the Formal Model 59
4.4.3. Application Components and Deployment Levels 61
4.4.4. Deployment Level Hierarchy Description 62
4.4.5. Structure of Dimensions, Groups, and Tenants 63
4.4.6. Description of Deployment Models 65
4.4.7. Completeness of Deployment Constraints 66
4.4.8. Definition of the Deployment Information 67
4.5. Prototypical Realization 68
4.5.1. Description of Realization Approach 69
4.5.2. Implementation of the Model using OWL 70
4.5.3. Model checking 73
4.5.4. Extraction of Deployment Information 75
4.6. Evaluation 77
4.6.1. Example Environment 78
4.6.2. Scenario Definition 80
4.6.3. Analysis of Results 81
4.7. Summary 84

5. Computation of a Valid and Optimal Deployment 87
5.1. Fundamental Concepts and Approaches III 89
5.1.1. Additional Concepts of Graph Theory 89
5.1.2. Concepts in Complexity Theory 90
5.1.3. Introduction of Graph Coloring and Clique Cover 91

5.2. Formal Definition of the Deployment Problems 96
5.2.1. Problem Analysis 96
5.2.2. Definition of the General Deployment Problem 100
5.2.3. Definition of the Elementary Deployment Problem 106
5.3. Analysis of Elementary Deployment Problem 109
5.3.1. Definition of a Solution as a Set of Clique Covers 109
5.3.2. Minimal Clique Cover on High Level 110
5.3.3. Minimal Clique Cover on Low Level 113
5.3.4. Minimal Clique Cover on Any Level 116
5.3.5. Complexity of the Elementary Problem 117
5.3.6. Introduction of Heuristics 118
5.4. Analysis of General Deployment Problem 120
5.4.1. Complexity of the General Problem 120
5.4.2. Generalization of Elementary Problem 121
5.4.3. Introduction of Heuristics 121
5.5. Evaluation 124
5.5.1. Experimental Comparison of Heuristics 125
5.5.2. Application on Running Example 127
5.5.3. General Efficiency of Resource Utilization 128
5.6. Summary 131

6. Case Study: ERP-System as Mixed-Tenancy Cloud Service 133
6.1. Conceptual Design of a Mixed-Tenancy Platform 135
6.1.1. Problem 1: Automated Deployment 135
6.1.2. Problem 2: Communication Mechanism 137
6.2. Selection of Cloud Application 143
6.2.1. An ERP-System as SaaS Offering - Requirements 143
6.2.2. Analysis of Available Open Source ERP-Systems 144
6.2.3. Introduction of OpenERP 145
6.3. Definition of OpenERP Scenario 146
6.3.1. Introduction of Component-based OpenERP 147
6.3.2. Selection of Application Components 149
6.3.3. Definition of Deployment Levels 150
6.3.4. Creation of Example Scenario 151
6.4. Realization of Deployment Platform for OpenERP 152
6.4.1. Problem 1: Automatic Deployment 153
6.4.2. Problem 2: Communication Mechanism 155
6.5. Evaluation and Discussion of Results 156
6.5.1. Evaluation of Mixed-Tenancy OpenERP 157
6.5.2. Discussion of Resource Utilization 158
6.6. Summary 160

7. Summary and Conclusion 163
7.1. Discussion of Results 163
7.1.1. Contributions 164
7.1.2. Limitations and Future Work 166

7.2. Summary 170

A. Details related to Case Study 171

A.1. Description of Application Components 171
A.2. Evaluated Test Cases 178

B. Mixed-Tenancy: Changes over Time 191

B.1. Problem Analysis 191
B.2. Computation of a Migration Plan 194
B.2.1. Abstraction on a Problem of Graphs Theory 195
B.2.2. Complexity of Finding a Migration Plan 196
B.2.3. Computation of a Migration Plan 197
B.3. Conclusion 199

C. Optimization considering limited Resource Availability 201

C.1. Problem Analysis 201
C.2. Introduction of Bin-Packaging (with Conflicts) 204
C.3. Conclusion 205

Bibliography 207

List of Figures

1.1. Illustration of Research Methodology 6

2.1. Illustration of Single and Multi-Tenancy (adapted from [CC06]) . . 13
2.2. Architectural Overview 16
2.3. Description of Problem using Entity Relationship Model 17
2.4. Prototype Architectural Overview 22

3.1. Results of Mapping Study [Sil+13] 37

4.1. Definition of Deployment Models using Venn Diagrams 51
4.2. Description of Dimensions and Group Structure 52
4.3. Description of the Utilization Process 53
4.4. Description of the Deployment Constraint 56
4.5. Description Deployment Information Extraction 57
4.6. Description Model as UML Class Diagram (incomplete) 58
4.7. Realization of the Description Model using OWL (incomplete) . . . 71
4.8. Evaluation Scenario that Includes all Possibilities 78
4.9. Definition of Example Dimensions and Groups 79
4.10. Deployment Information of Scenario 3 83

5.1. Relationship between Clique and Independent set [KN09, pg. 58] . 89
5.2. Solving encountered Clique Cover by Coloring the inverse Graph . 95
5.3. An Example of a Valid and Optimal Deployment 97
5.4. Extremes how Customers may share Units of one Deployment Level 99
5.5. Description of Example's Solution according to formal Definition . 100
5.6. Example why $\sim_{dl_i}$ is Required 103
5.7. Example not utilizing a minimal Clique Cover on highest Level . . 110
5.8. Corresponding Graphs for $n = 2$ (Lemma 2) 112
5.9. Example utilizing minimal Clique Cover on lowest Level 113
5.10. Alternative Solution - no minimal Clique Cover on lowest Level . . 114
5.11. Corresponding Graphs for $n = 4$ (Lemma 3) 115
5.12. Corresponding Graphs for $n = 4$ (Lemma 4) 117
5.13. Results of Example Introduced in Section 4.6 129

6.1. Automatic Deployment for two Deployment Levels [Rei13] 136
6.2. Overview of Communication Problem [Rei13] 137

6.3. Overview of Pattern to Tackle the Communication Problem [Rei+14]. 139
6.4. Overview of the Connector Pattern [Rei13] 142
6.5. Example Installation of OpenERP with 4 Modules [Rup13b] 147
6.6. Mixed-Tenancy Installation of OpenERP with 4 Modules [Rup13b] 148
6.7. Deployment Level of the investigated Scenario [Mor14] 150

B.1. Trivial Example Migration 194
B.2. A more complex Example Migration 195
B.3. Optimal Migration Plan for more Complex Example 197
B.4. Vertices vs. Tenants (two Deployment Levels) [OY13] 198
B.5. Real-Time A* (Lookahead depth = 1) vs. Brute-Force [OY13] 199

C.1. An Example for a Valid and Optimal Deployment 203

List of Tables

4.1. Introduction of used Notation 45
4.2. Mapping between Stacks and Application Components 79
4.3. Mapping between Tenants and Groups 80
4.4. Deployment Constraints for Scenario 3 82

5.1. Experimental Comparison of Approaches for Elementary Problem 126
5.2. Experimental Comparison of Approaches for General Problem . . 127

6.1. Summary of presented Patterns 141
6.2. Details about Relevant Applications 145
6.3. Resource Demand for Mixed-Tenancy and Single-Tenancy scenario (Required storage in mega bytes) 159

A.1. Successfully Executed Test Cases 190

C.1. Example Resource Limitation 203
C.2. Example Resource Demand of Tenant 203
C.3. Example Resource Demand of Unit 204

List of Definitions

Definition 1. Software/Application Vendor 10
Definition 2. Operator 10
Definition 3. Customer or Tenant 11
Definition 4. User 11
Definition 5. Application Component 11
Definition 6. Single-Tenancy 13
Definition 7. Multi-Tenancy 13
Definition 8. Mixed-Tenancy 15
Definition 9. Deployment Constraint 15
Definition 10. Valid and Optimal Deployment 18
Definition 11. Deployment Information 19
Definition 12. Deployment Configuration 20
Definition 13. Deployment Configuration Generator 22
Definition 14. Mixed-Tenancy Deployment Platform 22
Definition 15. Cloud Service Provider 23
Definition 16. Description Model 42
Definition 17. Utilization Process 42
Definition 18. Deployment Level 47
Definition 19. Deployment Unit 47
Definition 20. Deployment Model 49
Definition 21. Group 51
Definition 22. Dimension 51
Definition 23. Virtual Tenant 52
Definition 24. Deployment Computation Algorithm 88
Definition 25. General Mixed-Tenancy Deployment Problem 96
Definition 26. Elementary Mixed-Tenancy Deployment Problem 96
Definition 27. Resource 98
Definition 28. Resource Demand of Tenant 98
Definition 29. Resource Demand of Unit 98
Definition 30. Execution Engine 135
Definition 31. Initial Deployment 192
Definition 32. Target Deployment 192
Definition 33. Migration Operations 193
Definition 34. Migration Plan 193
Definition 35. Migration Graph 196

Chapter 1 Introduction

1.1. Motivation 1
1.2. Goals and Contributions of this Work 2
1.3. Structure and Content 3
1.4. Research Methodology 4

Computer Science has only two jobs -
one is creating complexity, the other is hiding it.

Georg O. Strawn

1.1 Motivation

Multi and Single-Tenancy are two architectural paradigms that describe how Customers use instances of an application. The basic idea of Multi-Tenancy is that multiple Customers (Tenants) use the same instance of an application at the same time [CC06]. In order to be able to do that, the Multi-Tenancy application needs to be able to separate the data of different Tenants from each other at run-time [CC06; BZ10; SR11; WA11]. The major advantage of Multi-Tenancy is that it allows Operators to utilize infrastructure resources most efficiently and allows them to focus maintenance on just one version of the application [CC06; BZ10; SR11]. According to recent studies, the major disadvantage of Multi-Tenancy is, however, the threat of information breach. This means that data of one Customer is accessible by another which may be a competitor [Clo13b; SK11; Sri+12; OWA13a; Sil+13; KMMG13]. Due to this many Customers have hesitations towards the usage of Multi-Tenancy applications since they fear that their data is disclosed due to a system malfunction, system error, or aggressive actions by individual Users [PB10; SK11].

Single-Tenancy, on the other hand, is a paradigm according to which each Customer gets its own, designated instance of the application and there is no sharing between them [CC06; BZ10]. The advantage of this is that the security threats of Multi-Tenancy do not apply [Clo13b; SK11]. For Operators, on the other hand, Single-Tenancy requires more utilization of infrastructure resources [CC06; SK11; SR11].

This work proposes a novel approach that allows reaching a reasonable trade-off between the Operators' need for exploitation of economies of scale and the Customers' fear of privacy threats. This approach is called Mixed-Tenancy since it is a hybrid approach between Single and Multi-Tenancy. It allows Customers to specify their constraints about if or with which other Tenants they would feel comfortable to share parts of an application and their underlying infrastructure. Based on all these Deployment Constraints expressed by Customers, a deployment of the application is computed that uses minimal resources but still applying to all Constraints. According to the computed deployment, the Operator may deploy and provision the entire software system to provide it to Customers.

1.2 Goals and Contributions of this Work

Based on the previous discussion, it is possible to express the general research question of this work as follows.

> *How can Customers' demand for expressing their Deployment Constraints and Operator's demand for having minimal cost both be satisfied at the same time?*

In order to answer this question, this work proposes the aforementioned Mixed-Tenancy approach. Thereby, this work compiles the following contributions.

Mixed-Tenancy Description Model A formal definition of a model will be given that allows capturing Customers' Deployment Constraints. Based on this model, Customers will be able to express if or with whom they want to share instances of Application Components. They will be able to do that not just by describing explicitly with which other Tenants they feel or feel not comfortable to share infrastructure, but also by using more abstract means of description. Such means would be, for example, defining Deployment Constraints based on geographic regions (e.g. no sharing with Customers from Asia), data protection law (e.g. only with companies following the European data security acts), or industries (e.g. no sharing with competitors). Further, it is possible to combine multiple Deployment Constraints to be more powerful (e.g. sharing only with companies from Europe but not with competitors). However, the Description Model presented in this work will only serve as a generic framework that allows formulating these kinds of Deployment Constraints. According to which Dimensions Customers may define their Constraints is fully customizable for the Operator. This allows for tuning the possible Deployment Constraints to the specific application that shall be offered following the Mixed-Tenancy paradigm. Furthermore, Deployment Constraints shall not just be expressible for the Application Components but also for the underlying infrastructure stack. This allows Customers to express that they may not feel comfortable to share application instances with other Tenants, but sharing the underlying virtual machine (a software based computer) may be acceptable. A prototypical realization of this model is created using technologies from the area of Semantic Web.

Deployment Computation Algorithm Based on all Constraints that are given by Customers, the Deployment Computation Algorithm will compute a deployment that applies to all given Deployment Constraints – the computed deployment is Valid. Furthermore, in order to satisfy the Operators' need for low operational cost, the algorithm must produce a deployment that not just covers all Deployment Constraints, but also only uses a minimal amount of resources. The deployment is Valid and Optimal. This way both stakeholders' concerns, those of the Customers as well as those of the Operator, are addressed. As part of this work a formal definition of the optimization problem will be given in order to have appropriate means for analyzing the problem. Based on this, the complexity of the problem will be analyzed. Further, possible algorithmic approaches will be presented and discussed.

Mixed-Tenancy Deployment Platform Once the Valid and Optimal Deployment has been computed, a platform is required that can deploy and operate a Multi-Tenancy application according to the Mixed-Tenancy paradigm. This work will discuss and provide solutions for multiple challenges that need to be tackled to create such a platform. These are the automatic deployment of a Multi-Tenancy application according to a computed deployment and ways of establishing communication between different Application Components deployed according to the Mixed-Tenancy. Furthermore, a prototypical realization of a Mixed-Tenancy Deployment Platform will be described, which will be able to provision a real-world Multi-Tenancy enterprise resource planning system as a Mixed-Tenancy application.

In order to evaluate the contributions of this work, all developed artifacts will also be realized prototypically. This way realizability will be analyzed. Furthermore, all concepts developed in this work will be applied to a case study. It will represent a scenario, how it would be found in a realistic environment. In fact, an application that is used by many companies in industry will be altered the way that it is deployable in a Mixed-Tenancy setting. This way the case study will demonstrate real-world applicability.

1.3 Structure and Content

This work is structured as follows. **Chapter 2**: *Towards Mixed-Tenancy - A Problem Analysis* , analyzes the problems to be addressed by this work in detail. In order to do that, the chapter starts with an introduction of the problem domain and relevant paradigms. Concepts, relevant for this work, will be defined in order to have a solid foundation to analyze the problems that shall be tackled by this work.

The outputs of this chapter are the research questions that shall be addressed. Furthermore, cloud computing is introduced as one possible area of application for the presented approach of this work.

Based on the problem definition, **Chapter 3**: *State of the Art* continues this work by doing an analysis of the state of the art. It, therefore, will analyze existing approaches to tackle the challenges of this work, and point out how the approach in literature differs from the approach presented in this work. The output of this chapter will be the conclusion that the approach of this work is novel and has not been presented in literature earlier.

Chapter 4: *Capturing Customers' Deployment Constraints* will present the first contribution of this work, the Mixed-Tenancy Description Model, which allows to capture Customers' Deployment Constraints. After introducing relevant concepts for the chapter, it goes on by first conducting a requirements analysis to find out what the Description Model needs to be able to capture. After that a high level process will be defined that demonstrates how the Description Model may be utilized for a specific Multi-Tenancy application. Based on this, the chapter continues by introducing the Deployment Model by defining it in a formal and complete way. The chapter concludes with the description of a prototypical implementation using technologies from the area of Semantic Web.

Based on the Description Model, it is possible to extract the Deployment Information. This information captures which Tenants may share resources and mark the input for **Chapter 5**: *Computation of a Valid and Optimal Deployment* . In this chapter it is discussed how a Valid and Optimal Deployment may be computed. It does that by first discussing the optimization problem and defining it formally. Further, the complexity of the problem will be analyzed. Based on the formal definition, characteristics of a Valid and Optimal Deployment will be analyzed. In addition, solution approaches will be discussed and compared that are supposed to tackle the problem. The most suitable ones will be selected in order to be used as a Deployment Computation Algorithm.

Chapter 6: *Case Study: ERP-System as Mixed-Tenancy Cloud Service* will analyze the real-world applicability of the Mixed-Tenancy approach by conducting a case study. Thus, a scenario will be discussed where a Cloud Service Provider wishes to offer a Software-as-a-Service application that follows the Mixed-Tenancy paradigm. In order to achieve that goal, the chapter will describe the design and realization of a Mixed-Tenancy Deployment Platform. Furthermore, an ERP-System will be selected that may be provisioned by this platform.

Chapter 7: *Summary and Conclusion* will conclude this work by summarizing the results. Furthermore, the contributions with respect to the proposed research questions will be summarized and the limitations of the presented approach will be discussed.

1.4 Research Methodology

According to [MG95], [GVR02], and [GRV04] research in software engineering may be categorized according to four research approaches. These are formulative, evaluative, descriptive, and developmental. A major part of the research conducted by this work may be categorized as developmental, as it deals with *"generating knowledge for explaining or solving general problems."* [MG95]. Further, other parts of this work, especially those related to the case study and the Deployment Platform may be considered evaluative. The evaluative research approach is *"involving methodologies that employ scientific methods, and usually consisting of [...] model generation or observation followed by hypothesis generation and testing"* [MG95].

Furthermore, [GVR02] and [GRV04] aggregated several research methods that are commonly used in the area of software engineering. Those that were utilized in the creation of this work are the following.

Conceptual Analysis This method's purpose is to gain knowledge by analyzing a phenomena conceptually [GVR02].

Conceptual Analysis – Mathematical A conceptual analysis is performed by utilizing mathematical techniques [GVR02].

Mathematical Proof This method's draws conclusions by proving them mathematically [GVR02].

Concept Implementation The primary purpose of this method is to provide a proof of concept by implementation [GVR02].

Laboratory Experiment – Software This method's purpose is to draw conclusions by conducting experiments [GVR02].

Case Study A case study *"is an empirical method aimed at in investigating contemporary phenomena in their context"* [RH09].

In fact, [GVR02] showed that the methods conceptual analysis, conceptual analysis – mathematical, and concept implementation are the dominant methods used in the software engineering area. This work does not make an exception to that end, as it also relies heavily on those three methods.

However, a detailed overview of how the results of this work were compiled is given by Figure 1.1. Please note that the major artifacts that are going to be created within this work are marked green. This work starts with conducting an analysis of the problems and challenges that arise from the Mixed-Tenancy approach. The outputs of this are three research questions that shall be the scope of this work. They deal with the capturing of Customer Deployment Constraints, computation of a Valid and Optimal Deployment, and applicability of the Mixed-Tenancy approach in real-world scenarios.

These research questions will be addressed in sequential order starting with the first. It deals with the creation of a model that allows the capturing of Deployment Constraints, called Description Model. Thus, the first thing to be done is to determine the necessary expressiveness of the Description Model. Expressiveness refers to the Deployment Constraints the Description Model is supposed

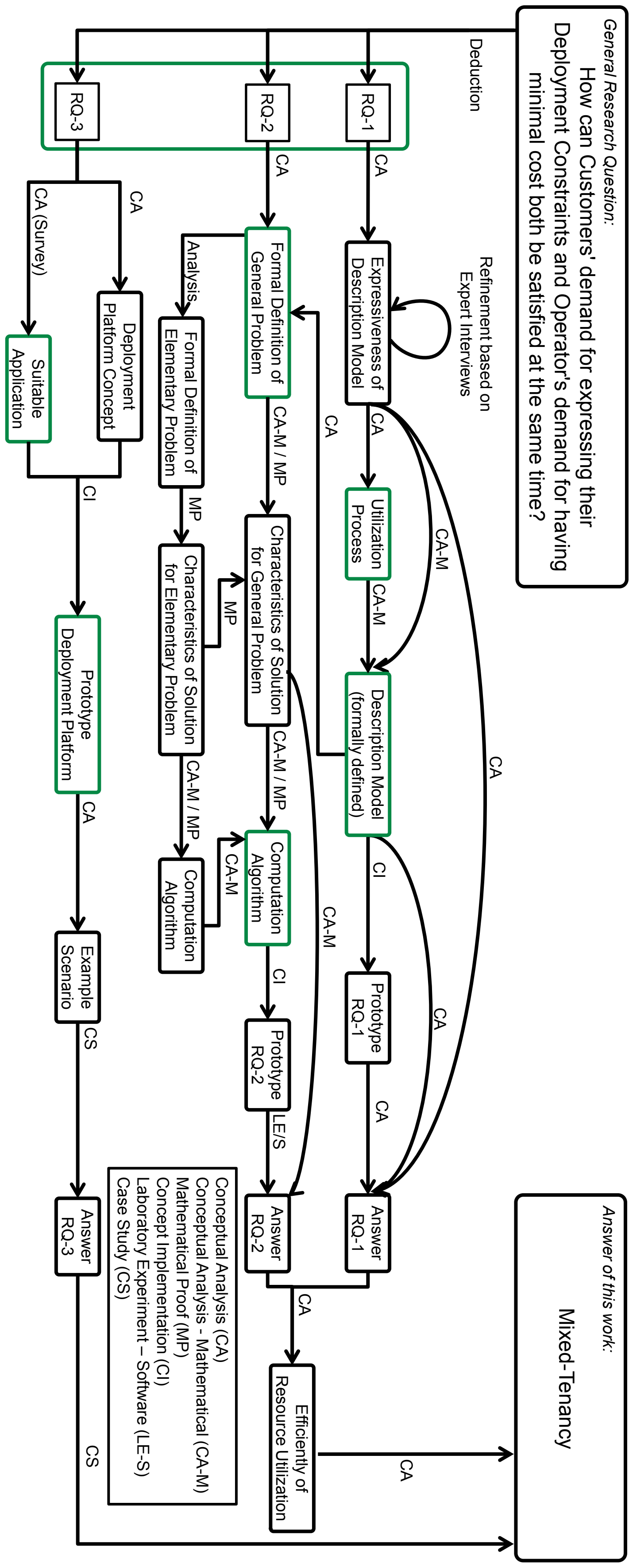
General Research Question:
How can Customers' demand for expressing their Deployment Constraints and Operator's demand for having minimal cost both be satisfied at the same time?
Deduction
RQ-1
RQ-2
RQ-3
CA
CA
CA
CA (Survey)
Refinement based on Expert Interviews
Expressiveness of Description Model
Formal Definition of General Problem
Analysis
Formal Definition of Elementary Problem
Deployment Platform Concept
Suitable Application
CA
CA
CA-M
CA-M / MP
MP
MP
CI
Utilization Process
CA-M
Description Model (formally defined)
Characteristics of Solution for General Problem
Characteristics of Solution for Elementary Problem
Prototype Deployment Platform
CA
CA-M / MP
CA-M / MP
CA-M
CA-M
Computation Algorithm
Computation Algorithm
CI
CI
Example Scenario
CS
Prototype RQ-1
Prototype RQ-2
CA
CA
LE/S
Answer RQ-1
Answer RQ-2
Answer RQ-3
CA
Efficiently of Resource Utilization
CA
CS
Conceptual Analysis (CA)
Conceptual Analysis - Mathematical (CA-M)
Mathematical Proof (MP)
Concept Implementation (CI)
Laboratory Experimentation – Software (LE-S)
Case Study (CS)
Answer of this work:
Mixed-Tenancy

Figure 1.1.: Illustration of Research Methodology

to be able to capture. The necessary expressiveness was defined using an iterative process. This process started by first coming up with an early version of the model based on literature and experience. This early version was presented to several experts from both academia and industry. Based on their feedback, the Description Model was extended and altered. It has gone through several such iterations[1]. Once the expressiveness of the model is defined, the model itself will be developed in a formal, unambiguous, and technology-independent way. For the formal definition first-order logic is used since it allows to create unambiguous models that represent a situation encountered by computer science professionals [Hut11]. The output of this is the first artifact called *Description Model* illustrated in Figure 1.1. Based on this formal definition a prototypical implementation is presented. Using this prototype it is possible to evaluate if the Description Model allows capturing all aspects that it was supposed to. This evaluation is done using an example that encapsulates the full range of expressible Deployment Constraints. Thus, the method of evaluation can be considered to be a concept implementation and conceptual analysis.

The second research question deals with the creation of the *Deployment Computation Algorithm*. The first step to tackle this question is to create a formal and unambiguous definition of the problem to be addressed. It will be called General Mixed-Tenancy Deployment Problem. The input for this definition is both the research question and the formal description of the *Description Model*. Based on the formal problem definition, it is the first goal to analyze the structure of a Valid and Optimal Deployment and the problem's complexity. This is done formally by using the conceptual analysis – mathematical and mathematical proof methods. Furthermore, algorithmic approaches are proposed and their performance guarantee is analyzed and proven. Due to the complexity of the problem, these analyses are conducted not based on the General Problem but a simplified version called the Elementary Problem. Conclusions gained from the Elementary Problem are afterwards generalized for the General problem. The entire analysis is based on the formalization of the optimization problem, thus, the method can be considered as conceptual analysis – mathematical and mathematical proof. However, to further elaborate on the performance of the proposed algorithmic approaches especially in real-world scenarios, implementations will be created. These implementations of the two approaches are afterwards compared based on randomly generated problem instances. This comparison uses the laboratory experiment – software research method.

The third research question deals with an analysis of the applicability of the Mixed-Tenancy approach in real-world. This research question was investigated by conducting a case study. The case study of this work is performed by investigating a case in which the Mixed-Tenancy approach is applied to deploy a real-world application in the area of cloud computing. For this it was necessary to identify a suitable application and develop a Deployment Platform. Using both, it is

[1]As an example, [RARV12] describes an early version of the model. Later in [RWV13] a more sophisticated version is presented. Between both there are many iterations and expert interviews.

possible to analyze real-world applicability of the Mixed-Tenancy approach.

Based on the results produced for all three research questions, it will be concluded whether Mixed-Tenancy is a valid approach to tackle the aforementioned general research question. For this conclusion, the resource utilization of the Mixed-Tenancy approach will be analyzed.

Chapter 2
Towards Mixed-Tenancy - A Problem Analysis

2.1. Fundamental Concepts and Approaches I 10
2.1.1. Relevant Stakeholders 10
2.1.2. Component-Based Software 11
2.1.3. Single-Tenancy and Multi-Tenancy 12
2.2. Scope of this Work 14
2.2.1. Introduction of Mixed-Tenancy 15
2.2.2. Challenges of Mixed-Tenancy 17
2.2.3. Research Questions of this Work 19
2.2.4. Limitations of Scope 20
2.2.5. Deployment Configuration Generator 22
2.3. Application of Mixed-Tenancy in Cloud Computing . . . 23
2.3.1. Introduction to Cloud Computing 23
2.3.2. Application of Mixed-Tenancy in Cloud Computing . . 25
2.4. Summary 26

Every problem is a gift - without problems
we would not grow.

Anthony Robbins

It is the goal of this chapter to perform a problem analysis and clearly state what problems are to be tackled by this work. In order to achieve this goal the chapter produces the following artifact as output:

Research Questions The research questions clearly state the challenges addressed by this work. Once defined the research questions will serve as a point of reference to the remainder of this work.

In order to create the research questions, the chapter starts by introducing the fundamental concepts and approaches that make up the problem domain of this work. One of these approaches, that are to be introduced, is Multi-Tenancy.

Multi-Tenancy bears certain opportunities for Operators. However, Customers have hesitations towards using Multi-Tenancy applications. This contradiction will be discussed in detail and an approach is proposed that strives to dissolve this contradiction. Thus, this chapter will define the scope of the study by introducing Mixed-Tenancy as the novel approach of this work. This is done by first introducing the approach briefly and then analyzing the challenges that arise from it. Based on these discussions the research questions of this work are defined. To conclude the chapter, cloud computing will be introduced as one possible area of application for the concept of Mixed-Tenancy. Cloud computing still bears certain security threats that may be tackled by introducing Mixed-Tenancy.

The chain of argument is structured as follows. Section 2.1 continues the chapter with an introduction of the fundamental concepts and approaches relevant for this chapter. Based on these, Section 2.2 defines the scope of the research that is conducted by this work. Further, in Section 2.3 cloud computing is introduced since the contributions of this work are applicable very well in this particular area of business. The chapter concludes with a summary (Section 2.4) of the most important points made by this chapter.

2.1 Fundamental Concepts and Approaches I

This section introduces fundamental concepts and approaches on which this chapter builds upon. It does so by first introducing the stakeholders that are involved with the problem domain. Based on this, the concept of component-based software is introduced as an architectural paradigm for the applications that may be provided using the novel approach of this work. Further, the concepts of Single-Tenancy and Multi-Tenancy are described and distinguished. Each subsection will introduce definitions, the remainder of this work builds upon.

2.1.1 Relevant Stakeholders

The first step towards the introduction of the problem domain is to define the different stakeholders that are involved with it. For the remainder of this work we distinguish the stakeholders *Application Vendor*, *Operator*, *Customer/Tenant*, and *User*. The same or similar categorization of stakeholders into these four roles has been done before by research that has been conducted in the same problem domain (e.g. [Sch+12b; Mie10; Feh+14]). Those are the definitions of the four stakeholders used by this work.

Definition 1 (Software/Application Vendor) An *Application or Software Vendor* is a company that develops an application and provides it to the market.

Definition 2 (Operator) The *Operator* deploys, runs, and maintains an application on rented or owned hardware infrastructure. Based on this, it is provided to the *Customer*.

Definition 3 (Customer or Tenant) The entity called *Customer* or *Tenant* is a company that consumes an application provided by the Operator. The term *Customer* is used to refer to this entity from a business perspective. The technical term for the same entity is *Tenant*. For the remainder of this work, both terms will be used interchangeable as they refer to the same entity.

Definition 4 (User) A *User* is a person or employee that has access to an application and, thus, interacts with it. Each *User* belongs to or is employed by exactly one *Customer/Tenant*.

The terms that have just been defined, were introduced in a way as if they were representing disjoint entities. In reality, however, it may be possible that a single entity may manifest itself as multiple different stakeholders. An example would be a company that is Application Vendor as well as Operator at the same time and, thus, not just develops an application but also deploys, runs and maintains the application. Furthermore, it would be possible that a particular company is also using the application internally and, therefore, also becomes a Customer.

For the remainder of this work the stakeholders will be used as if they were referring to separate entities. This is done, solely, in order to reduce complexity and increase readability. It does not limit the applicability of this work's contributions.

2.1.2 Component-Based Software

The fundamental idea of a component-based software system is that the system is a composition of a multitude of components, rather than being a huge monolithic system. According to [Szy02, pg.3] a software component can be defined as *"executable unit of independent production, acquisition, and deployment that can be composed into a functioning system"*. Interfaces are used by components to define which functionality and properties they provide to and require from the environment and/or the surrounding system.

The primary advantages of component-based systems are that they often benefit of enhanced adaptability, scalability, and maintainability [Szy02]. Further, another advantage of component-based systems is that a software system may utilize both, components that are custom-made from scratch and those that are bought from third-party vendors. This allows Software Vendors to increase the efficiency of development.

Component-based software is one of the fundamental approaches this work builds upon. The reason for this is that it is the goal of this work to allow Tenants to express Constraints, about with which other Tenants they are willing to share certain parts of the same application. These parts are the Application Components that compose the application. Based on the definition of a software component introduced by [Szy02], the following is a definition of what is referred to as an Application Component within this work.

Definition 5 (Application Component) An *Application Component* is defined as a building block of an application that encapsulates atomic functionality.

All functionality and properties they provide to and require from other *Application Components* shall be captured by a described interface, through which all interaction flows. Furthermore, it is highly important for this work, that *Application Components* may be deployed independently from each other.

In fact, especially the focus on having the possibility for independent deployment is of particular importance for this work. This is due to the fact that one of the primary challenges is that Application Components are deployed multiple times, in order to be used by different Tenants. This is only possible if they may be separated from each other. The challenges involved actually applying the Mixed-Tenancy approach to a given application will be discussed in detail in Section 6.1.

2.1.3 Single-Tenancy and Multi-Tenancy

This section's purpose is to discuss and define different styles of how Customers may be served through the instance of an Application Component. This is done by first introducing maturity models well-known from literature and then based on them defining the relevant models for this work.

In general, it is possible that a single instance of an Application Component may be used by only one Customer, or is shared by multiple Customers at the same time. To elaborate on this in more detail, [CC06] defines four levels of maturity for sharing instances. The levels build upon each other, which means that every level is distinguished from the previous one by the addition of one additional characteristic. The following describes the four levels.

Level I: Ad Hoc/Custom At this level each Customer uses their own designated instance that is specifically customized to their needs. Customization is achieved through multiple different code implementations that are specific to provide certain sets of features.

Level II: Configurable Similar to the first level, the second level also deploys designated instances for every Customer. In contrast, however, all instances are created based on the same code base. Customizations that are made for specific Customers are purely done on a configuration level.

Level III: Configurable, Multi-Tenant-Efficient At this level of maturity, the Operator only hosts a single instance for multiple Customers. The different Customers' data, however, are isolated from each other. The goal of that is not to give any indication to Customers that they use a shared instance, but promote the impression that the instance is designated to them. Furthermore, still all Customers shall be able to have specific customizations made for them, providing unique feature sets and a unique user experience.

Level IV: Scalable, Configurable, Multi-Tenant Efficient Similar to the previous level, instances are being shared by multiple Customers and customization is purely done by configuration. However, this time there is not just one instance but multiple that are simultaneously used by multiple

Customers. There is no fixed assignment of Tenants to specific instances but the assignment is made based on the availability of resources. The advantage of using a load-balanced farm is improved scalability, as it is possible to increase and decrease the number of instances according to Customers' demand.

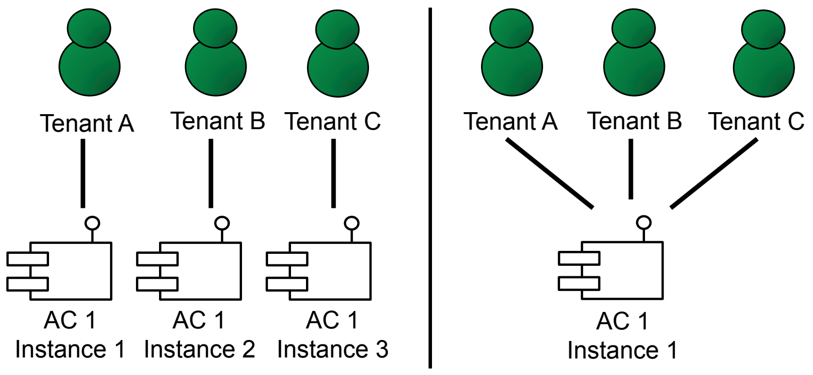

Figure 2.1.: Illustration of Single and Multi-Tenancy (adapted from [CC06])

Based on this categorization, for the remainder of this work the terms Single-Tenancy and Multi-Tenancy will be used according to the following definitions.

Definition 6 (Single-Tenancy) The term *Single-Tenancy* refers to an Application Component that is being deployed individually for every Customer, but all Customers share the same code. Customizations are made purely on a configuration level. This definition applies to maturity level two (Configurable).

Definition 7 (Multi-Tenancy) *Multi-Tenancy* means that a single instance of an Application Component serves multiple Customers. Due to configuration each Customer may have a unique feature set and user experience. Furthermore, different Customers are treated isolated from each other in order to hide the fact that they are actually accessing shared instances. The definition applies to maturity level three (Configurable, Multi-Tenant-Efficient).

These two definitions are also illustrated by Figure 2.1.

For Operators, the Multi-Tenancy model is more desirable. This is due to the fact, that it offers multiple opportunities for saving cost by exploring economies of scale. The major point is that resources are shared between multiple Tenants. Thus, Operators will require less infrastructure to offer an application compared to the Single-Tenancy model. But the required resources are not the only opportunity to save cost, the Operator may also minimize the effort necessary for maintaining a high number of instances [BZ10].

However, Multi-Tenancy comes with a price tag attached. For the Software Vendor, an application following this approach is more complex to develop and maintain, since the entire application needs to be built in the way that it can handle the data of multiple Tenants simultaneously by still enforcing isolation between them. Thus, this leads to higher effort and cost [BZ10]. The extra complexity needs to be paid for by the Operator. Due to the increased complexity of application development, a threat for Customers arises. If Multi-Tenancy is not properly implemented, it may be possible that the isolation between Tenants fails. This might lead to a situation in which one Tenant can access the data of another [ZSLB09; PB10; SK11; Sri+12; Clo13b].

The benefit of Single-Tenancy is that these kinds of problems cannot occur. This is due to the fact that in an environment in which instances are not shared, additional mechanisms can be implemented to separate Tenants from each other. For example, it would be possible to separate Customers by deploying them on different virtual machines on the same physical server and installing firewall rules that separate their networks. Furthermore, it may even be possible to completely separate Tenants physically, using separate networks and servers. From an Operator's perspective, however, Single-Tenancy is less desirable since it bears almost no opportunity for exploitation of economies of scale [CCW06; BZ10; GKS13].

In order to conclude the discussion about Single-Tenancy and Multi-Tenancy it is necessary to address one more aspect. All Application Components are built and deployed on a stack of infrastructure that they utilize. This infrastructure stack may vary from Application Component to Application Component. For example, it would be possible that an Application Component written in Java would require a Java Application Server that is run on a Linux Server in a Virtual Machine. Another Application Component created using Python might require a Python run time environment that is run on Windows and a physical Server.

In the maturity levels defined by [CC06], and the definitions given by this work it was only stated that an instance of an Application Component is shared or not. There has been no mention so far about the underlying infrastructure stack. For the Multi-Tenancy model, where instances are shared, it is quite obvious that the underlying infrastructure must be shared as well. For Single-Tenancy, however, this statement may not be made that easy. It may be possible that Single-Tenancy only applies to the instances Application Component, but the underlying infrastructure or only parts of it are shared by different Customers. It is up to the Operator of an application to determine for which parts of the infrastructure stack Single-Tenancy shall be offered.

In the novel approach presented by this work, this choice is made by the Customers themselves.

2.2 Scope of this Work

Based on the introduction of the fundamental concepts and approaches provided by the last section, it is this section's purpose to define the scope of this work. In order to do so, the section starts with an introduction of Mixed-Tenancy, the novel approach proposed by this work (Section 2.2.1). It continues by analyzing challenges (Section 2.2.2) that arise from the approach and defines the scope of this work by defining three research questions that are to be tackled (Section

2.2.3). To define the scope of this research even further, Section 2.2.4 discusses issues that will not be addressed by this work. To conclude the section, a high-level overview of the target architecture will be given that will serve as a point of reference for the reader (Section 2.2.5).

2.2.1 Introduction of Mixed-Tenancy

In the introduction of Single-Tenancy and Multi-Tenancy (Section 2.1.3) it was discussed that both bear advantages and disadvantages for different stakeholders. From a Customer's perspective it is desirable not to share an instance using Multi-Tenancy, as security and privacy may be decreased. From an Operator's perspective, however, it is very desirable to have as many Customers as possible share the same application instance. This is due to the fact that it decreases operational cost as well as maintenance (only a single instance needs to be maintained) and allows thereby exploiting economies of scale.

The novel approach that is presented in this work tries to satisfy both stakeholders' demands by finding a hybrid way between Single-Tenancy and Multi-Tenancy. It does that by allowing Customers to express specific constraints that define if or with whom they feel comfortable to share resources, but still use application instances most efficiently within the boundaries of what is acceptable for the Customers. This approach is referred to as Mixed-Tenancy.

Definition 8 (Mixed-Tenancy) *Mixed-Tenancy* is a hybrid approach between Single-Tenancy and Multi-Tenancy. It allows Customers to express their constraints about if or with which other Tenants they are willing to share resources. Based on these constraints, a deployment is computed that uses minimal resources in order to satisfy Operator's concerns. *Mixed-Tenancy* is the novel approach of this work.

Within this work the basic idea of the approach is applied to component-based Multi-Tenancy applications. Thus, it is possible that multiple instances of the same Application Component might be deployed at the same time. This allows Customers to express their constraints not just for the entire application at once, but they can do that for every individual Application Component. These constraints are referred to as Deployment Constraints within this work.

Definition 9 (Deployment Constraint) A *Deployment Constraint* is a Customer's description of if or with which other Tenants resources may be shared. It is necessary that a Tenant gives a number of *Deployment Constraints* for the deployment of an entire application.

To illustrate this further Figure 2.2 depicts an example. In this example there are four Tenants (A to D) and seven Application Components (1 to 7) an application is composed of. The four Customers shall have the ability to express their Deployment Constraints about if or with whom they wish to share instances of the seven Application Components. The seven Components handle data that are different in sensitivity. For this example Component 1 handles very sensitive/confidential

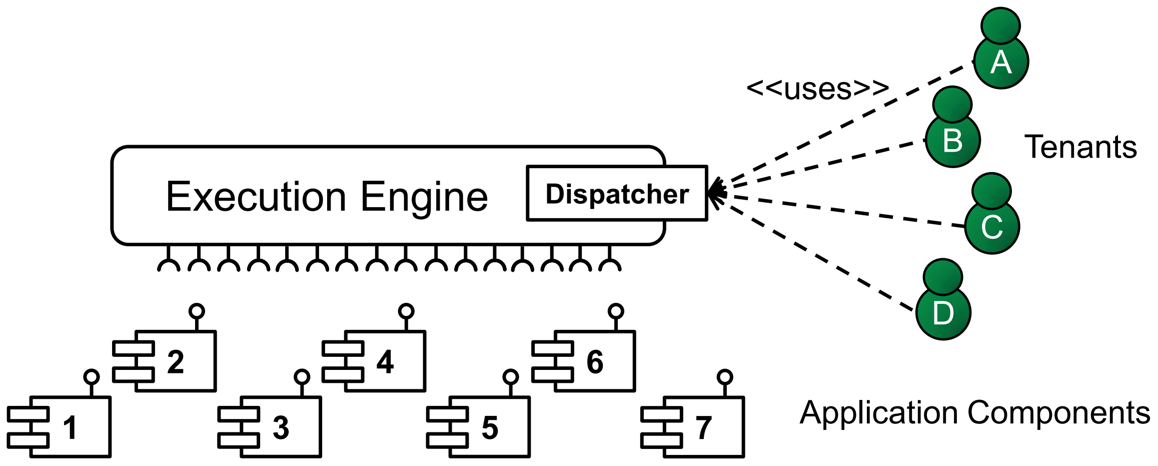

Figure 2.2.: Architectural Overview

data and Component 7 only handles data that is public knowledge. Sensitivity of Application Component 2 to 6 are different degrees between those extremes. The following may be examples of Deployment Constraints that shall be possible to express.

- Tenant A is very concerned about security, thus, they demand that they do not share instances with other Customers at all.
- Tenant B is concerned about security issues, thus, they demand that the very sensitive Components (Application Component 1-2) are deployed designated to them. For the Application Components handling data that is categorized as middle sensitivity (Application Component 3-5), they have the demand that they only share with companies that come from Europe but which are not competitors. And finally, for the low-sensitivity Application Components (Application Component 6-7) they are willing to share with companies that are following data protection laws similar or equal to those of the European Union.
- Tenant C is only little concerned with security, thus, they demand to only share high sensitivity Application Components (Application Component 1-2) with Tenants from the EU or North America. For the middle sensitivity Application Components (Application Component 3-5) they demand to share only with companies that are not competitors. And for the low sensitivity Application Components (Application Component 6-7) they do not have Deployment Constraints and are willing to share with any Customers.
- Tenant D is not concerned about security at all. They trust in the application's implementation of Multi-Tenancy and are very price conscious. This is why they do not have any Deployment Constraints towards the sharing of instances.

In the example that was presented, Customers' Deployment Constraints were limited to the sharing of instances of Application Component. However, the approach proposed by this work shall go further than this. It shall be possible for Customers to express not just Deployment Constraints for the sharing of Application Components but also for the underlying infrastructure. An example for this would be that the very security conscious Customer A requires, as stated, designated instances of Application Components but would be willing to share the underlying Virtual Machines and physical Servers with other Customers from Europe but not with competitors.

2.2.2 Challenges of Mixed-Tenancy

The previous section gave an introduction to the Mixed-Tenancy approach. Based on this, it is this section's goal to elaborate this even further by discussing the challenges the Mixed-Tenancy approach bears. These challenges need to be overcome in order to be able to apply the approach. In order to analyze the challenges Figure 2.3 gives a generic overview of some of the challenges. As

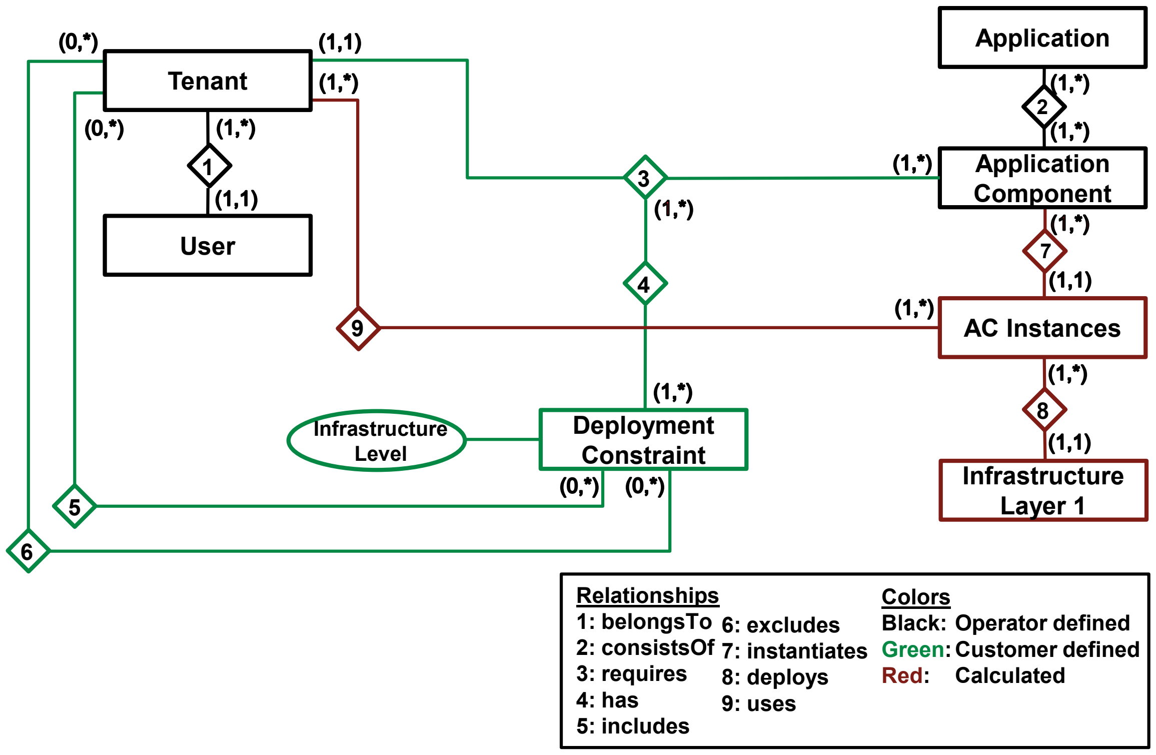

Figure 2.3.: Description of Problem using Entity Relationship Model

previously mentioned for this work, a Tenant represents a company that has a number of Users that use the application. In Figure 2.3 this is indicated by the entities *Tenant* and *User* as well as the relationship *belongsTo* (relationship 1).

In addition, it has previously been discussed that the approach that is utilized in this work is based on the paradigm of component-based software. Due to this, the application that is offered to Customers, is composed of a number of Application Components. In the figure this is indicated by the entities *Application* and *Application Component* as well as the *consistsOf*-relationship (relationship 2).

All entities and relations that have just been introduced need to be defined by the Operator. The Operator needs to define which application shall be offered, what Application Components it is composed of, and which Tenants want to use it.

Based on the Operator's definition of the environment, it is possible for Tenants to express their Deployment Constraints (using the elements marked red in the figure). This is indicated in Figure 2.3 by the entity *Deployment Constraint*, the property *Infrastructure Level*, as well as the relationships *requires* (relationship 9), *has* (relationship 4), *includes* (relationship 5), and *excludes* (relationship 6). All this means that a Tenant has to give a Deployment Constraint for every Application Component they use. In fact, there may be multiple Deployment Constraints, one per Infrastructure Layer (e.g. Application Component Instances and Virtual Machines). Moreover, a Deployment Constraint may also define with which other Tenants resources shall be shared or not. The *include* and *exclude* relations do just that. Furthermore, it is possible to express that a Customer does not want to share a resource at all or has no Deployment Constraints with whom they share. This may be covered by excluding or including all Tenants. Please note that a much more sophisticated approach to capturing Customer Deployment Constraints will be introduced as part of this work (in Chapter 4).

Expressing Customer Deployment Constraints using this model can be considered the first challenge of Mixed-Tenancy. This is due to the fact that the ability to capture those constraints is a precondition for all following steps.

Based on the Customers' description, the next challenge arises. In Section 2.2.1 it was stated that Mixed-Tenancy is an approach to address both, the Customers' and the Operator's concerns. For the Operator it is crucial that a minimum number of resources are being utilized in order to allow for exploitation of economies of scale. Thus, the second challenge is to find a Valid and Optimal Deployment based on the description given by the Customers.

Definition 10 (Valid and Optimal Deployment) A deployment is valid if it applies to all *Deployment Constraints* that were defined by all Customers. It is optimal if it causes only minimal cost by utilizing only a optimal number of instances of the Application Component and the underlying infrastructure layers[1].

The structure of a Valid and Optimal Deployment is indicated by the items marked red in Figure 2.3. In the figure there are two entities representing infrastructure layers, *Application Component Instances* and *Infrastructure Layer 1*[2]. The number of Units of each infrastructure layer needs to be determined by an algorithm that determines the Valid and Optimal Deployment (*instantiates*-relationship). Furthermore, it is necessary to determine which Tenant uses which

[1]A more specific and formalized definition will be given in Chapter 5. Furthermore, the relationship between the cost of a deployment and its required number of Application Component Instances and the underlying infrastructure layers will be elaborated.

[2]Please note that there shall be additional infrastructure layers possible. This is going to be further elaborated in Chapter 4.

instances of Application Components (*uses*-relationship) and which Unit of infrastructure layer 1 deploys which Application Component Instance (*deploys*-relationship).

All previous steps are necessary to gather the requirements for a Deployment and compute it. However, once a Valid and Optimal Deployment has been found, it is necessary to realize it. This means that a given application is automatically provisioned on a platform according to the defined Deployment.

2.2.3 Research Questions of this Work

The previous two sections (Sections 2.2.1 and 2.2.2) introduced the reader to the general idea of the Mixed-Tenancy approach and discussed challenges that arise from this approach. Concluding from the problem analysis, the general question of this research is formulated as follows:

> *How can Customers' demand for expressing their Deployment Constraints and Operator's demand for having minimal cost both be satisfied at the same time?*

Based on this general question it is possible to derive the following, first research question:

Research Question 1 (RQ-1) How can Customers' Deployment Constraints towards Mixed-Tenancy be described?

- **Research Question 1.1 (RQ-1.1)** What shall be capturable in order to be able to describe Deployment Constraints?
- **Research Question 1.2 (RQ-1.2)** How is the description being performed by the participating stakeholder?
- **Research Question 1.3 (RQ-1.3)** What does a model look like that is capable of capturing Customers' Deployment Constraints?

Within this work these questions are to be tackled by the creation of a description approach that utilizes a Description Model that allows capturing Deployment Constraints. Based on all Deployment Constraints given by all Customers, it is necessary to extract the information about which Tenants may share which resources. For the remainder of this work this information is referred to as Deployment Information.

Definition 11 (Deployment Information) *Deployment Information* is created based on all Deployment Constraints given by Customers. It describes which Tenants may share which resources on an aggregated level.

The Deployment Information is the input for the next research question.

Research Question 2 (RQ-2) How can a Valid and Optimal Deployment be found in a fast[3] way?

[3]As commonly used in literature, fast is used in the way that it means in polynomial time [Wan06; JM08]

It is the goal of this work to create an algorithm that is capable of computing such a deployment based on the input provided by previous step. Further, Valid and Optimal Deployment is captured by the Deployment Configuration.

Definition 12 (Deployment Configuration) The *Deployment Configuration* is a transfer document that is generated by the Deployment Configuration Generator (which will be introduced in Section 2.2.5). It describes a Deployment.

In order to determine the applicability of the Mixed-Tenancy approach it would be possible to implement a simple dummy application that may be deployed in according to the Mixed-Tenancy approach. However, this work goes one step further and tries to evaluate the applicability of the approach to existing component-based Multi-Tenancy applications. Thus, the research questions concerning this issue may be expressed by the following.

Research Question 3 (RQ-3) Is it possible to apply the Mixed-Tenancy approach to existing composite Multi-Tenancy applications?

- **Research Question 3.1 (RQ-3.1)** How can a platform realizing Mixed-Tenancy be created?

- **Research Question 3.2 (RQ-3.2)** Does Mixed-Tenancy allow an Operator to use resources more efficiently as with Single-Tenancy?

These research questions shall be addressed by conducting a case study (Chapter 6). Within this case study, an existing suitable application used in industry is identified and it is migrated to a Mixed-Tenancy platform. Based on the case study, it will be possible to gain conclusions for which kind of applications the Mixed-Tenancy approach is realizable.

2.2.4 Limitations of Scope

Based on the definition of research questions of this work, it is this subsection's goal to point out challenges that will not be addressed by this work[4]. This serves the goal to describe the scope of this work in more detail.

Changes over Time Once an application has been deployed according to Deployment Constraints given by Customers, it is very likely that this deployment has to change over time. Triggers for this may be changes of the Customer base (new Customers subscribe or existing Customers unsubscribe), direct changes of existing Customers' Deployment Constraints (e.g. an existing Customer changes the previously expressed Deployment Constraints), or indirect changes of existing Customers' Deployment Constraints (e.g. a Customer extends their area of business by penetrating a new market - they may become a competitor to other Tenants). However, these kinds of changes, that require changing an existing Deployment, will not be covered by this work. This work is limited to creating Mixed-Tenancy for a first-time deployment.

[4]In Chapter 3 related approaches will be presented that tackle some of these challenges.

Functional Variability Functional variability is the variability an application offers with respect to features. It will be discussed in Section 3.1.2 more thoroughly. However, this type of variability will not be addressed by this work. The scope of this work does not stand in contrast to approaches that allow introducing functional variability. Some of the approaches may be taken from the area of Software Product Lines. As part of this work's research, [RA11] analyzed opportunities to do that.

Collaboration Another opportunity that arises from Mixed-Tenancy is collaboration between Tenants. Since Tenants may express with whom they wish to share an instance of an Application Component, it would be possible to enable collaboration between those Tenants by allowing them to access the same data. This could be achieved by deactivating the isolation between Tenants for this instance. However, this would require that the Application Component has the functional variability to allow for such sharing. But since functional variability is out of scope for this work, so is the opportunity for collaboration. As part of this work, however, the opportunity has previously been discussed in [RARV12].

Deployment Variability In reality it is often the case that an application offers some kind of variability with respect to how it may be deployed. This kind of variability is not considered in this work. For this work it is assumed, that for every Application Component there is only one possible infrastructure stack that requires to be deployed properly. This is done in order to decrease complexity. If one Application Components would have multiple different infrastructure stacks they may rely on, there is a higher chance that infrastructure may be shared by different Application Components. Algorithms to perform such a mapping may be reduced to the festival problem and have been worked on by related work (e.g. [FLM10]).

Implementation of additional Security Measures Mixed-Tenancy has been introduced as an approach to increase security of Multi-Tenancy systems by separating Tenants where Customers do not feel comfortable to share infrastructure. The reason that this increases security is that it opens the opportunity for deployment of additional security measures. Concerning this issue, this work will not give a general comprehensive analysis of available and beneficial measures. This is due to the fact that these measures would be very specific to a given environment (e.g. infrastructure stack, used hypervisor for virtualization).

Non-Functional Requirements Considering the Customers' ability to express their requirements, this work is only limited to the expression of Deployment Constraints. Other non-functional requirements (e.g. Service Level Agreements) are not covered by this work. There is related work with respect to non-functional requirements and Multi-Tenancy applications (e.g. [FLM10; Feh09]). However, the novel approach of this work is the definition of Deployment Constraints.

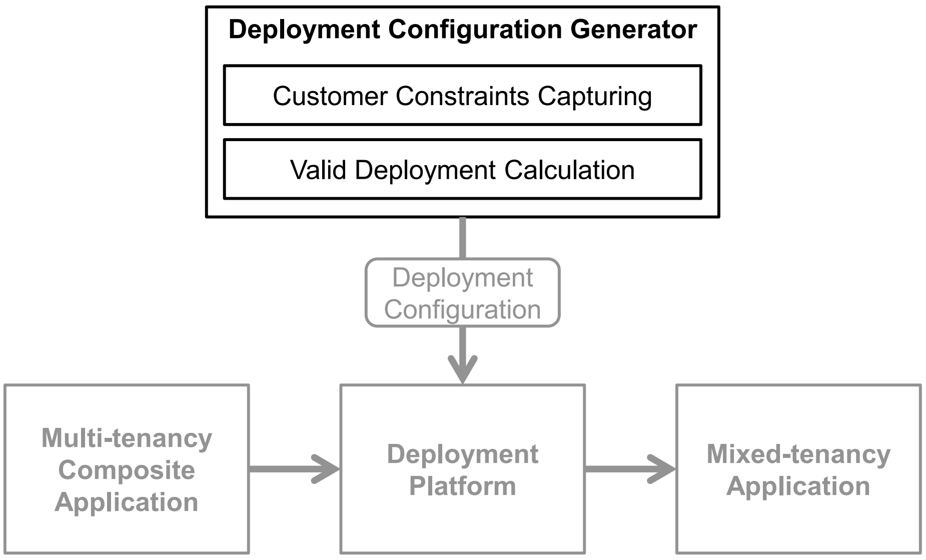
Deployment Configuration Generator
Customer Constraints Capturing
Valid Deployment Calculation
Deployment Configuration
Multi-tenancy Composite Application
Deployment Platform
Mixed-tenancy Application

Figure 2.4.: Prototype Architectural Overview

All the challenges that have just been introduced may be seen as opportunities for future research. Thus, they will be mentioned again in the final chapter discussing opportunities for future research (Section 7.1.2).

2.2.5 Deployment Configuration Generator

All concepts that are developed in this work will be realized by implementing a prototype. Figure 2.4 illustrates an abstract overview of it. The system represented in black is the Deployment Configuration Generator.

Definition 13 (Deployment Configuration Generator) The *Deployment Configuration Generator* is the prototypical implementation of the contributions of this work. It allows capturing Customers' Deployment Constraints and computes a Valid and Optimal Deployment. The output of it is the Deployment Configuration.

The Deployment Configuration Generator consists of two sub-components called Customer Constraint Capturing and Valid Deployment Calculation. Each of these corresponds to one of the previously defined research questions (*Customer Constraint Capturing* - RQ-1 and *Valid Deployment Calculation*- RQ-2).

The grays part of Figure 2.4 contribute to research question RQ-3. All of them belong to the platform that realizes Mixed-Tenancy.

Definition 14 (Mixed-Tenancy Deployment Platform) The *Mixed-Tenancy Deployment Platform* receives the Deployment Configuration as input and, based on it, creates a Mixed-Tenancy deployment of a Multi-Tenancy application and provisions it.

These parts were created specifically for the case study presented by Chapter 6 and will be discussed in detail.

2.3 Application of Mixed-Tenancy in Cloud Computing

This section introduces the reader to cloud computing as one possible area of the application for the novel approach of this work. This is of particular importance since the case study, conducted in Chapter 6, will demonstrate how the approach may be used to provision Mixed-Tenancy cloud applications.

This section lays out the groundwork for this by first introducing cloud computing and then discussing why this Mixed-Tenancy may be beneficial for a Cloud Service Provider and Customers.

2.3.1 Introduction to Cloud Computing

Over the previous couple of years cloud computing has been introduced as a change of paradigm how IT will be managed in the future. This 21st century's model of computing anticipates a significant transformation of the computing industry [Arm+09; Buy+09]. The model promises to no longer require service Users to invest in new infrastructure but receive their computing services as a 5th utility (after water, electricity, gas, and telephony). As a side-effect it eliminates the risk for Users running into problems when operating their existing already very complex infrastructure or adding new components to it. Instead, Customers pay Cloud Service Providers only for the IT resources based on their usage.

Definition 15 (Cloud Service Provider) A *Cloud Service Provider* is a company that offers Cloud Services to their Customer. Every *Cloud Service Provider* is also an Operator of a cloud service. In addition, the *Cloud Service Provider* provides the application to Customers following the cloud paradigm.

The advantage of cloud computing for a *Cloud Service Provider* is that due to shared resource pooling, they have the chance to exploit economies of scale.

When cloud computing became a buzzword, there were many definitions available that each tried to grasp the term (e.g. [Buy+09; Arm+09]). However, today the most commonly recognized definition is the one published by the National Institute of Standards and Technology (NIST) in 2011 [MG11].

It defines cloud computing as follows.

> *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. "* [MG11]

Furthermore, the NIST defines five essential characteristics, three service models, and four deployment models which are relevant to cloud computing. The characteristics are [MG11]:

On-demand Self-Service Cloud services are provided to Customers without further human interaction.

Broad Network Access Services are provided over standard networks that promote use of heterogeneous platforms.

Resource Pooling The Cloud Service Providers resources are pooled to serve multiple Customers at the same time. This allows the Provider to exploit economies of scale. This may be achieved through Multi-Tenancy.

Rapid Elasticity For the service consumers the availability of resources seem unlimited since resources are provisioned according to the current demand.

Measured Service Resource usage is monitored, controlled, and reported in order to provide transparency for both the provider and consumer.

In addition, the following three service models are distinguished [MG11].

Infrastructure as a Service (also known as IaaS) The Cloud Service Provider providing IaaS manages a large pool of computing resources, such as storing and processing capacity. Customers use parts of this pool and are charged based on their usage.

A well-known example for this IaaS is Amazon's platform Amazon Elastic Compute Cloud (Amazon EC2) [Ama13].

Platform as a Service (also known as PaaS) This service model adds more value to the capabilities of IaaS by adding one additional abstraction layer. Instead of supplying a virtualized infrastructure, PaaS provides a computing platform and/or solution stack as a service on which Customers can deploy and execute their applications. Thus, Customers do not have to deal with the cost and complexity of buying and managing the underlying hardware and software layers. The sizing of the hardware resources of the PaaS platform is done in an elastic scalable way in order to generally provide as much computing resources as the Customer demands.

A well-known example is the Google App Engine [Goo13].

Software as a Service (also known as SaaS) This is the service model with the highest capability and service value. Instead of provisioning an infrastructure on which Customers can build their applications, SaaS adds another abstraction layer and delivers entire applications to the Customer over the Internet. These applications are commonly web-based and Users access them by making use of a standard web-browser. SaaS providers usually offer these applications based on a pay-per-use fee. Customers, on the other hand, do not have to be concerned with the installation, operation, update etc. of the application or the infrastructure being used to host the application. SaaS applications profit greatly from an environment that adjusts to changing demands.

One of the more famous examples of an SaaS application is the Customer Relationship Management System SalesForce.com [sal13].

Finally, according to the NIST definition the deployment models are [MG11]:

Private Cloud This deployment model provides an exclusive cloud for every single Customer. The cloud may be operated by the organization itself or a third party and may exist on-premise or off-premise. If the private cloud is operated by a third party off-premise, it is similar to classic outsourcing.

Community Cloud This model allows sharing the infrastructure within a distinct group of organizations. This cloud is usually set up to the specific needs and requirements of this group. It could be managed by the organizations themselves or by a third party.

Public Cloud Using this deployment model a cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services. It is hosted at the site of a Cloud Service Provider. This model is especially suitable for Customers with a need for a cheap price and a highly standardized production environment. Therefore, Customers must accept low data security and data privacy levels.

Hybrid Cloud The cloud is composed of two or more infrastructures following different models (private, community, or public).

2.3.2 Application of Mixed-Tenancy in Cloud Computing

As discussed in the previous section one of the major benefits for the Cloud Service Provider to offer cloud services, lies in the chance of exploiting economies of scale. Multi-Tenancy, on the other hand, is a great architectural approach to allow just that. It allows that multiple Customers share the same Application Component and the underlying infrastructure. This is why SaaS-applications heavily rely on Multi-Tenancy [ZSLB09; SK11; SKP13].

Due to this, multiple, maybe even competing Customers, share the same instances of SaaS-applications and the underlying infrastructure. However, there are security risks involved with doing so. These were analyzed by the Cloud Security Alliance in the beginning of 2013.

The Cloud Security Alliance is a not-for-profit organization with a mission to promote the use of best practices for providing security assurance within cloud computing, and to provide education on the uses of cloud computing to help secure all other forms of computing [Clo13a].

In the beginning of 2013 the Cloud Security Alliance released [Clo13b], which is a report that analyzed the top security threats of cloud computing. This report is based on a survey of industry experts to compile professional opinion on the greatest vulnerabilities of cloud computing. The result of the report were nine critical threats for cloud security, that were ranked according to their severity[5]. In this report the following number one threat in cloud computing was listed which may be tackled by the Mixed-Tenancy approach.

[5]Starting from the most sever threat the identified threats are: data breaches, data loss, account or service traffic hijacking, insecure interfaces and APIs, denial of service, malicious insiders, abuse of cloud services, insufficient due diligence, and shared technology vulnerabilities.

Data Breaches The most severe threat according to Cloud Security Alliance is the risk of data breach in a shared environment. The threat is that sensitive internal data placed at Cloud Service Provider might fall into the hands of competitors, for example due to flaws in the SaaS-application. It is quite remarkable that 91% of the surveyed experts consider this threat to be relevant. Furthermore, the threat is considered equally severe with respect to perceived risk and actual risk.

Please note that similar descriptions of this cloud threat may be found in other sources as well (e.g. [SK11; OWA13a; Sil+13; KMMG13]).

For SaaS-applications the threat of Data Breach may occur on multiple layers of the application's infrastructure stack. One certain danger is the implementation of Multi-Tenancy [SK11; KMMG13]. This is due to the fact that Multi-Tenancy needs to be implemented by the developers throughout the application [OWA13b]. However, these problems may not just occur on the application layer but also on underlying infrastructure layers. In November 2012, for example, a mixed team from research and academia succeeded in extracting private cryptographic keys being used in other, neighboring virtual machines on the same physical server [ZJRR12].

Applying this work's contributions to the area of cloud computing comes quite naturally when considering the threat that has just been introduced. As discussed in Section 2.2, the Mixed-Tenancy approach allows Customers to define their constraints if or with whom they wish to share infrastructure. Thus, Customers may choose to define Deployment Constraints for those parts of SaaS-applications that handle very sensitive data, the way that they do not share infrastructure at all or do only share with a smaller group of Tenants. This will give the Cloud Service Provider the chance to implement further techniques to ensure security (e.g. on a network level). For those parts that do not handle sensitive data, Customers may choose to share infrastructure with other Tenants and, thereby, gain monetary benefits.

When categorizing the Mixed-Tenancy approach according to the definition of deployment models given by the NIST (introduced in Section 2.3.1), the Customer shall have the ability to choose between the different deployment models on Application Component level and the underlying infrastructure. It is up to the Customer to pick a private, public, or community model. In Chapter 4 a sophisticated model will be introduced that allows Customers to express their Deployment Constraints in a very detailed level. However, in the presented scenario infrastructure is always hosted at a Cloud Service Provider's site.

In Chapter 6 a case study will be conducted that analyzes the applicability of the Mixed-Tenancy approach in cloud computing.

2.4 Summary

It was the goal of this chapter to perform a problem analysis and clearly state what problems will be tackled by this work. In order to do this the chapter started by introducing paradigms and concepts that are relevant to the problem domain.

Thus, the Application Vendor, Operator, and Customer/Tenant were introduced as relevant stakeholders. Furthermore, the concept of component-based software was discussed since this work deals with composite applications. The concepts of Single-Tenancy and Multi-Tenancy were introduced as architectural approaches how application instances may be used by Tenants. It was discussed that Multi-Tenancy bears the threat of data breach. Thus, Tenants are often hesitant to use Multi-Tenancy application. On the other hand, however, Multi-Tenancy allows Operators to exploit economies of scale and, thus, is a beneficial approach to decrease cost.

Based on this, Mixed-Tenancy was introduced as an approach that strives to satisfy Customers' needs for security and unwillingness to share resources, as well as the Operator's desire to utilize existing infrastructure as efficiently as possible. After the problems related to Mixed-Tenancy had been discussed in detail, the following general research question was defined.

> *How can Customers' demand for expressing their Deployment Constraints and Operator's demand for having minimal cost both be satisfied at the same time?*

Further, from this general research question, the following three research questions were derived.

Research Question 1 (RQ-1) How can Customers' Deployment Constraints towards Mixed-Tenancy be described?

Research Question 2 (RQ-2) How can a Valid and Optimal Deployment be found in a fast way?

Research Question 3 (RQ-3) Is it possible to apply the Mixed-Tenancy approach to existing composite Multi-Tenancy applications?

Furthermore, for RQ-1 and RQ-3 sub-questions were defined. In order to define the scope of this work even further, a detailed description was given about challenges that are out of scope for this work.

The chapter concluded with the introduction of cloud computing as a possible area of application for the Mixed-Tenancy approach. Cloud computing was introduced according to the well-accepted definition of the NIST [MG11]. Software-as-a-Service, one of the service models of cloud computing, deals with the provisioning of entire applications from the cloud. For these kinds of applications, Multi-Tenancy is an important paradigm for Cloud Service Providers to exploit economies of scale. However, according to recent studies, the risk of data breaches is the major threat of cloud computing today. Mixed-Tenancy may be utilized as a tool to counteract against these risks.

Chapter 3
State of the Art

3.1. Research related to Multi-Tenancy 29
3.1.1. Realization of Multi-Tenancy 30
3.1.2. Variability of Multi-Tenancy Applications 33
3.2. Related Approaches tackling Security Issue 36
3.2.1. Introduction of the Underlying Mapping Study 36
3.2.2. Related Work tackling Data Breach 37
3.3. Summary 39

If I have seen further it is only by standing on the shoulders of giants.

Sir Isaac Newton

In the previous chapter the scope of this work was analyzed and described. It is this chapter's purpose to describe the state of the art related to this work.

In order to do that, the chapter is separated into two major section. Section 3.1 discusses related work that addresses the realization and variability of Multi-Tenancy applications.

In Section 2.3.2 the major security threats of cloud computing were identified for which the Mixed-Tenancy approach may be beneficial. Based on this, Section 3.2 introduces related approaches that allow to address this issue as well. This section is based on a mapping study that identified many papers addressing security threats in cloud computing. The chapter is concluded by a summary given by Section 3.3.

3.1 Research related to Multi-Tenancy

In this section, research related to Multi-Tenancy is going to be presented. In order to do that, Subsection 3.1.1 discusses how Multi-Tenancy may be realized. Based on this, Subsection 3.1.2 discusses variability of Multi-Tenancy applications, and introduces approaches to realize them.

3.1.1 Realization of Multi-Tenancy

In Section 2.1.3 the two terms Single and Multi-Tenancy were introduced. Building upon this, it is this section's purpose to elaborate further on the realization of Multi-Tenancy.

So far it has been defined that Multi-Tenancy allows an Application Component so serve multiple Customers at the same time. Therefore, the Application Components isolate the data of different Tenants from each other. Up to this point, however, no insight on the actual realization of Multi-Tenancy has been provided. A first step towards doing this, is to report on how Customers' data may be separated. [CCW06] gives an overview by introducing three distinct approaches for creating data architectures to realize Multi-Tenancy. These are the following.

Separated Databases The simplest approach for data isolation is storing Tenants' data in separate databases. In general, computing resources and application instances are shared between all Tenants, but the Tenants' data remains logically isolated from each other. Meta-data associated to each database describes which Tenant uses it. Further, access is restricted through the database security mechanisms. Thus, it is possible to prevent Tenants from accidentally or maliciously accessing other Tenants' data. The advantage of this data model is that it is easy to extend to specific Customer needs, restoring backups for a specific Tenant may be done very easily, and it offers added security. Cost, hardware and maintenance, however, are higher than for the other alternatives since the number of Tenants that can be served by a database server is limited to the number of databases it supports.

Shared Database, Separate Schemas The next approach is to store multiple Tenants' data in the same database but in separate schemas. Once a Tenant first subscribes to use a given application, it is necessary to create a set of database tables that are associated to this Tenant. Again, it is possible to restrict access to those tables only to those Users that belong to the authorized Tenant. This approach offers a moderate degree of logical data isolation for security-conscious Tenants, and supports a larger number of Tenants per database server. The most significant drawback of this approach is that Tenants' data is harder to backup and restore since it is not possible to restore a backup of an entire database. This would overwrite other Tenants' data. Overall this approach will result in lower cost for the Operator at the first approach.

Shared Database, Shared Schema The last approach's idea is to have a shared database and a shared schema used by multiple Tenants. Thus, a given table will include records from multiple Tenants. Individual records are associated to Tenants by a separate column (e.g. *TenantID*) in each table. Of the three approaches presented, this one has the lowest demand for hardware as it allows a database server to serve the largest number of Tenants (this is especially true for applications that rely on a high number of tables). On the other hand, however, this approach requires additional development effort to establish security mechanisms that ensure that a Tenant's data is never accessed by another Tenant, even in the event of unexpected bugs or attacks.

Obviously, it is extremely important for Multi-Tenancy applications that every aspect of the application is secure and data is only accessed by the authorized Tenants. Thus, [CCW06] introduces four security patterns.

Trusted Database Connection There are two methods commonly used to allow individual Users access different tables, views, queries, etc. These are called *impersonation* and *trusted subsystem*. Following the *impersonation* access method, the database is set up to grant access to individual Users. Thus, if a User performs an action that requires interaction with the database, the application will initiate a connection as this User. In the *trusted subsystem* method, the application always connects to the database as itself. Thus, the connection is independent of the identity of the User. The database grants the application access to those database objects the application is allowed to access. This method requires that additional security for limiting Users' access on data is implemented in the application.

In Multi-Tenancy systems database access methods are a little bit more complicated. This is due to the distinction between Tenants and Users. This additional complexity may be tackled by a hybrid approach of *impersonation* and *trusted subsystem*. In this approach the accessing permissions are associated to the Tenants. If a User triggers a transaction with the database, the application creates a database connection as this Tenant. This way the database does not distinguish transactions caused by different Users of the same Tenant. Thus, it is necessary that the application checks and grants access to data of a Tenant based on Users.

Secure Database Tables If the earlier discussed approaches separate-database and separate-schema are utilized, it is possible to grant access to Tenants on a table level. This may be achieved through the GRANT SQL command.

Tenant View Filter If the shared schema approach is utilized, it is not possible to use the secure database table approach. In this case, however, it is possible to use SQL views. In SQL a view is virtual table defined by the result of a SELECT query. Using views it is possible to create a Tenant specific virtual table that only contains their data. Thus, it is possible to grant individual Tenants access to their rows in a given table, while preventing them to access other rows. None of the Tenants would get access to the source table - only accessing their data through the view.

Tenant Data Encryption An additional way to protect the data of Tenants placed in the database is encryption assigning a unique key to every Tenant. This way data will remain secure even if it falls into the wrong hands.

Based on these concepts as foundation, it is possible to introduce more related work towards the realization of Multi-Tenancy. Much of the surrounding related work is focused on the issues related to security of Multi-Tenancy.

In [BAT12] the specific risks in cloud computing due to Multi-Tenancy are explored. Further, measures to counteract on these risks are analyzed. As a conclusion the paper states that Multi-Tenancy indeed introduces unique security risks and their countermeasures fall in the following three categories: Governance, Control & Auditing Configuration, Design & Change Management, and Logical Security, Access Control & Encryption.

In [AGI12], TOSSMA is introduced. It is a Tenant oriented security management architecture, that allows Tenants to define, customize, and enforce their security requirements without having to go back to application developers for maintenance or security customizations. Security attributes that are considered by the approach are identity management, authentication, authorization, logging, and cryptography.

[SR11] proposes a multi-tenant data architecture that allows to express customizable privacy constraints. For this data encryption and information dissociation is utilizes. This approach is based on customization features of SaaS applications and the, previously introduced, shared database shared schema approach.

Besides these security related issues, there is other research being done. In [KM08], for example, a multi-tenant placement model is developed which decides for a new Tenant which server would best accommodate it. The decision is mainly done based on available hardware resources such as CPU and storage. The overall goal of the placement model is to achieve maximum cost savings without violating any requirements of service level agreements.

Further, [Aul+08] introduces a new schema-mapping technique for implementing Multi-Tenancy. It is called chunk folding. The basic idea is to partition logical tables into vertical partitioned chunks. These are folded together into different physical Multi-Tenancy tables and joined as needed. Using this technique, it is possible to have a database model that is more scalable than the previously introduced models.

In [Guo+07] a framework is provided that allows the design and implementation of high quality native Multi-Tenancy applications more efficiently. It does that by proposing a new programming model and framework to simplify and speed up the application development.

[MK11] discusses different options for introducing Multi-Tenancy to existing applications. This requires initial reengineering effort. Thus, the paper introduces a simple cost model for evaluating different options for implementing Multi-Tenancy depending on the application at hand. It thereby considers the cost for operation/life-cycle management, the expected number of Tenants and the expected period of use.

[Koz11] proposes the SPOSAD architectural style, which describes the components, connectors, and data elements of a multi-tenant architecture as well as constraints imposed on these elements. This paper describes the benefits of such an architecture and the trade-offs for the related design decisions.

[CSL09] proposes an approach that allows to monitor Multi-Tenancy applications as well as resource allocation based on service level agreements. The proposed architecture, thereby, considers that different Customers have different requirements towards the service level agreements and allows to monitor the quality per Tenant. This enables the Operators to react on abnormal status and dynamically adjust resource usage.

Furthermore, [MUTL09] and [Mie08] discuss how services in a process-aware service-oriented SaaS application can be deployed using different Multi-Tenancy patterns. In the context of this paper Multi-Tenancy patterns are whether a service is configurable or not and if it may host multiple Tenants at the same time. Even if this paper sounds quite similar to this work, it actually differs in many ways. First of all the research is focused on process-aware applications. In addition, the main contribution of this work, that Customers may actually express their Deployment Constraints, is not addressed at all.

An application offered as a service in the cloud is often configurable regarding non-functional qualities, such as location or availability. In [Feh09] and [FLM10] the challenge of resource allocation is addressed that arises if multiple Tenants wish to use use these services by expressing their individual requirements. The paper presented a framework that allows Operators to use distribution strategies as well as to define them on their own. Optimization may be performed regarding individually modeled resource and system properties. Again, although the paper is related to the scope of this work, differs however as it did not address the issue of Customer defined Deployment Constraints but focuses on other non-functional requirements.

The ZDNet article [Wai08] addresses the issue of different flavors of Multi-Tenancy, and the vendors that use it. In this article the term hybrid tenancy is mentioned as a lesser-degree Multi-Tenancy where Customers only share lower Levels of the infrastructure stack. The approach of this work may be seen as an addition to this article as it allows to create such hybrid tenancy systems by adding the notion of Customer specific Deployment Constraints.

3.1.2 Variability of Multi-Tenancy Applications

An earlier version of the following text has partially been published in [RA11].

For applications, or more generally speaking, information systems, there are two general types of flexibility that are important [GS06]. The first one is called flexibility in the pattern of use (*flexibility-to-use*) and refers to a manufacturing machine's ability to perform different operations [SS90; GS06]. For information systems this corresponds to the number of Users' requirements the information system can cover without requiring a major change. The second kind of flexibility is called *flexibility-to-change* and describes an application's extensibility and the effort necessary to implement a major change. The possibilities for this range from systems that cannot be changed at all, to systems that promote changes by including a large set of opportunities (e.g. due to a modular architecture).

In order to distinguish the two different kinds of flexibility, it is necessary to define what is meant by a *major change*. A major change of the information system can be defined as adjustments and modifications that require a fresh system setup, re-compilation, re-installation or re-testing [GS06]. Applying this to Multi-Tenancy applications, a major change would require the Operator to change the code of the application and redeploy it for all Tenants. The following compares the need for flexibility for both, Single-Tenancy and Multi-Tenancy systems.

In a Single-Tenancy environment, a Tenant may require to have multiple Users working on the same instance. Since the system instance is designated to serve only a single Tenant, it can be adapted according to this Tenant's needs. This adaptation may be done through customization capabilities that the application provides, but also by altering the source code. Which way of adaptation is used depends on the application's customization capability and on the Customer's needs. Customization might concern all levels of the application (e.g. data models, service/component realization, processes or user interfaces) [Nit09]. However, the adaptation is done prior to the application's deployment. In case the application shall be changed (e.g. due to changed requirements) a major change would be necessary. Thus, the main type of flexibility a Single-Tenancy application needs to provide is flexibility-to-change. Besides the flexibility-to-change capability, it is very desirable for a Single-Tenancy application to provide some degree of the flexibility-to-use since common applications may also offer the ability to be adapted while the application is deployed and running (e.g. selected language).

In contrast to Single-Tenancy applications, Multi-Tenancy applications require to have their flexibility of the category flexibility-to-use. This is due to the fact that it is not possible, in their case, to deploy an application instance specifically adapted to the needs of only one Customer. This concludes that the customization of an application needs to be done specifically for a Tenant instead of specifically for an application instance. If a Multi-Tenancy application is created that offers a very strong variability of the flexibility-to-use kind, it will behave differently, depending on the Tenant using it.

The challenge of introducing flexibility into Multi-Tenancy applications has been addressed in several related works. As an example, [Sun+08] discusses the configuration issues and challenges related to it. In addition, the paper proposes a competency model and a methodology framework that both aim to support Software-as-a-Service vendors in planning and evaluating their configuration and customizing strategies.

In [ADG02] and [DBV05] the configuration and customization of web services is discussed. However, in this work customization is only done in the flexibility-to-change phase.

In [Nit09] the issue of how to effectively and efficiently support configurability in Multi-Tenancy software is addressed and a Software-as-a-Service architecture to support configurability is proposed. In this work configurability data is implemented in XML format. In contrast to this paper's vision there is a strong focus on a single case study instead of a general method.

In [LHLP10] an approach is presented to describe variability for Software-as-a-Service applications. This approach can systematically describe variability points and their relationships, and assures the quality of the configuration inputs made by the Customers. This work focuses on the creation of descriptions of variability but not so much on the execution.

In [Zha+07] the authors propose a policy based Software-as-a-Service customization framework that is realized through a design-time tooling and a run-time environment. However, this work mainly focuses on the issues in service customization for a given set of requirements.

In research the approach of software product lines has proven to be valuable for the development of software products with a strong need for diversity [PBL10]. The basic idea is to apply the concept of large-scale production of goods tailored to individual Customers (mass customization) to software [PBL10]. The approach used to realize this starts by analyzing the required commonality and variability of a product line. Based on this, a platform is created from which the individual applications that suit specific Customer's needs are derived. However, creating an application using software product lines means that the output is an application instance that is specifically tailored, shipped, deployed and run for the Customer. Thus, categorizing this concept according to the discussed categorization of flexibility means that it is placed partially on the flexibility-to-change area. Nevertheless, in software product lines there is a big collection of techniques that allow to tackle this challenge which can be utilized in all phases of the application's creation (product architecture derivation, compilation, linking, and run-time) [SGB05]. In [SGB05] a total of 16 possible software product lines realization techniques were presented. Only six of them are actually usable in the Linking, and run-time phase of an application and, thus, useable for Multi-Tenancy applications. However, besides the realization techniques, software product lines also bring techniques to capture variability of software. In literature there are multiple approaches that apply these techniques to Multi-Tenancy.

In [Mie10] the author discusses bringing flexibility to the process layer of process-based, service-oriented Multi-Tenancy applications. This is done by creating variability descriptors that are transformed into a BPEL process model.

The different papers [Sch+12b], [SLW12], and [Sch+12a] propose an extended feature model to express variability of functionality and service qualities for Multi-Tenancy applications. These feature models allow different stakeholders to express their requirements towards the application. Further, a concept for dynamic configuration management to address the identified requirements is proposed. It is a staged configuration process that is capable of adding and removing stakeholders dynamically. That allows for reconfiguration of variants as stakeholders' objectives change.

In [Wan+11] the authors use a hypergraph-based service model to represent services and Multi-Tenancy applications. Based on these graphs, it is possible to represent hierarchical service and application structures from which Multi-Tenancy applications can be constructed that fulfill Customers' needs.

3.2 Related Approaches tackling Security Issue

In Subsection 2.3.2 a major security threat of cloud computing was introduced for which the Mixed-Tenancy approach is supposed to be beneficial. It is this section's purpose to present related work which discusses other approaches that were trying to tackle the same problem, and relate them to the approach of this work.

The section is based on a mapping study that was conducted in March 2013. Thus, the section starts with an introduction of this study (Subsection 3.2.1). Based on this, Subsection 3.2.2 will present related work regarding the security threat.

3.2.1 Introduction of the Underlying Mapping Study

In [Sil+13] a systematic mapping study was conducted to identify literature that deals with security threats in cloud computing. In this study the digital libraries Elsevier Scopus, IEEExplore, ACM Digital Library, SpringerLink, Science Direct, and Engineering Village were used to identify suitable literature. This was done by applying the following filter rule: {[(Noncompliance with security) OR (keywords for security)] AND [cloud computing solutions]} in the title or abstract or keywords in the article. This resulted in the following search string:

> ("flaw" OR "risk" OR "threat" OR "vulnerabilit*" OR"unsafe" OR "untrust") AND
> ("security" OR "safe" OR "trust") AND
> ("cloud" OR "multi-tenan*" OR "*aas" OR "* as a service" OR "* as-a-service")

Applying this search to all aforementioned sources reveals a total of 1011 publications. Due to the openness of the search string, many of these publications were actually not in the research context. Thus, a manual inspection was performed, where a superficial screening was used to filter the 1011 publication. This was done by applying the following criteria[1].

1. Inclusion Criteria
   - Security in cloud computing as the main theme.
   - The publication should have some relationship with one of the seven threats[2].
   - The publication should have a proposed solution.
2. Exclusion Criteria

[1]Please note that the following enumeration is a direct quote, taken from [Sil+13]

[2]Please note that [Sil+13] relies on the 2010 version of the report on threats in cloud computing by the Cloud Security Alliance. In contrast to the newer version, cited in Section 2.3.2, the 2010 version identified only seven instead of nine threats. However, the relevant threat, that may be addressed by this work's approach, data breaches, is found in both the 2010 and 2013 version.

- Duplication of publication.
- Journals not accessible online.
- Publications with related threat, but not active in cloud computing.
- Publications that only bring a revision or approach, without a proposal of solution.

3. Relevance Criteria
    - Papers with well-detailed solution proposal;
    - Papers that have some kind of proposal validation, with statistical data, experiment, etc;
    - Papers focused in fulfilling some compliance;

Based on these criteria, the number of papers was stripped down to 661. Figure

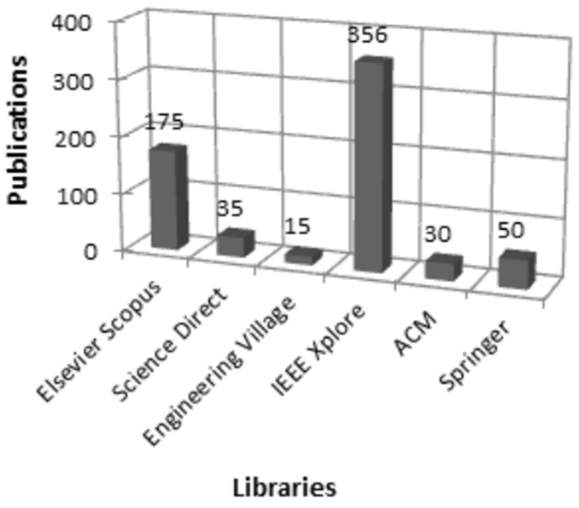

Figure 3.1.: Results of Mapping Study [Sil+13]

3.1 gives an overview where those publication were found. The identified 661 publications were categorized according to the security threats identified by the Cloud Security Alliance.

For this work, the authors of the mapping study kindly provided their database naming the 661 publications for the creation of this work. This database has been filtered for the publications dealing with the security threat, data breaches, addressed by this work. Those publications associated to the cloud computing threat, were analyzed with respect to their relevance for this work. The most relevant for this work are going to be presented in the next subsections.

3.2.2 Related Work tackling Data Breach

The mapping study [Sil+13] contained a total of 125 publications that address the issue of data breach. For this work they were analyzed and the following is an overview of the most relevant approaches for this work.

In [RTSS09] opportunities for information leakage are investigated. To be more precise, the authors investigate the Amazon EC2 as case study and show that it is possible to map the internal cloud infrastructure in order to predict where a target virtual machine will be deployed. Based on this information, it is possible to place a new virtual machine co-located to the target and attack the target virtual machine using cross-vm side-channel attacks to extract information.

Similar to the previous paper, [DM11] demonstrates, based on simulation, that cloud computing systems that use open source code could be subject to a simple malicious attack that degrades the availability of virtual machines. Further, the authors discuss how this attack leads to virtual machine leakage, thereby reducing the pool of resource available to Users.

In [LDPS10], the issues related to Windows guest cloud service resilience are analyzed. Based on this, the authors propose an architecture, called CReW (Cloud Resilience for Windows) that allows to transparently monitor guest Windows virtual machines and react on security breaches and system integrity violation in order to improve the dependability of Windows systems in cloud scenarios.

In [SMS10], the authors propose a new cloud based infrastructure which allows to differentiate between applications and data. Based on this differentiation, the authors introduce the concept of trusted data binding, enforcing policy usage on application over data sets with the aid of trusted hardware such as the trusted platform module. A prototypical implementation was provided in Amazon EC2 which allows software providers to upload software and data owners can search for algorithms and use them to be executed privately on their data sets.

Based on the assumption that Cloud Service Providers may be trusted, [Cha+11] defines a trust, security and privacy preserving infrastructure. Thereby, the authors open source software supports trust and reputation management, sticky policies with fine grained access control, privacy preserving delegation of authority, federated identity management, different levels of assurance and configurable audit trails.

In [Okt+12], the authors propose a framework that allows to distribute data and processing in a hybrid cloud. This is supposed to allow to meet the conflicting goals of performance, sensitive data disclosure risk and resource allocation cost. The solution was implemented as an add-on tool for an Hadoop and Hive based cloud computing infrastructure. In contrast to the research of this work, [Okt+12] focuses strongly on distributing workloads between different cloud types. This work's primary purpose was to allow Customers to define Deployment Constraints for the creation of hybrid and/or community clouds.

There is a lot of research that deals with the encryption of data within a cloud. As an example, both [Cao+11] and [Cao+14], both, address the issue of enabling an encrypted cloud data search service. In [Cao+14], the authors define and solve the problem of a privacy-preserving multi-keyword ranked search over encrypted data in cloud computing. Further, they establish a set of strict privacy requirements for such a secure cloud data utilization system.

Another work related to data encryption for the cloud is [PKZ11]. The authors propose to identify and encrypt all functionally encryptable data. This refers to all sensitive data that can be encrypted without limiting the functionality of the application on the cloud. For this purpose the authors present a tool called silverline that allows to automatically identify all functionally encryptable data, assign encryption keys to specific data subsets to decrease key management complexity, and establish transparent data access at the User's devices.

[LZL10] establishes a framework and supports a reference model for cloud computing by adapting the concept of insurances. This model guarantees service assurance, integrity and quality of service. It, therefore, uses quantitative or qualitative metrics to apply the basis of the business value and risk assessment. For this it applies calculations used for insurance premium and compensation calculation in order to calculate the failures of the services offered in cloud environment.

Cloud computing and virtualization becoming mainstream has caused a need for the ability to track data from creation to its current state. Thus, [Zha+12] introduces an approach for tracking end-to-end data provenance, a meta-data describing the derivation history of data. By utilizing and analyzing this provenance, it detects various data leakage threats and alerts administrators and data owners.

Similar to the previous paper, [AK12] also proposes an approach to track data. It thereby proposes an architecture or system which provides an intelligent track in privacy manager and risk manager to address privacy issues which rule the cloud environment.

3.3 Summary

This section introduced the current state of the art related to this work's scope. In order to do that, it started by introducing approaches and related work that are dedicated to the realization of Multi-Tenancy. Further, the matter of variability in Multi-Tenancy systems has been addressed. In order to do that it discussed two types of variability: flexibility-to-use and flexibility-to-change. It was stated that for Multi-Tenancy systems there is a strong focus on flexibility-to-use since it enables an application to behave differently depending on the Tenant that is using it.

The second part of this section presented related research that tackles security issues in Multi-Tenancy. More specifically, this related work addresses the major security issue addressed by this work: Data Breach. It did that based on a mapping study ([Sil+13]) that was found in literature. Thus, most relevant pieces of related work were introduced.

The conclusion gained from this section is that the approach introduced by this work is indeed novel and has not been presented in research earlier.

Chapter 4
Capturing Customers' Deployment Constraints

4.1. Fundamental Concepts and Approaches II 43
4.1.1. First-order Logic 44
4.1.2. Introduction of Graph Theory 44
4.2. Conceptual Analysis based on Requirements 46
4.2.1. Levels of Deployment 46
4.2.2. Deployment Models 49
4.2.3. Groups 51
4.2.4. Dimensions 51
4.2.5. Virtual Tenants 52
4.3. Process of Deployment Constraint definition 53
4.3.1. Customizing 53
4.3.2. Tenant Grouping 55
4.3.3. Constraint Definition 55
4.3.4. Deployment Information Extraction 56
4.4. Generic Mixed-Tenancy Description Model 58
4.4.1. Introduction of the Fundamental Idea 58
4.4.2. Foundations of the Formal Model 59
4.4.3. Application Components and Deployment Levels 61
4.4.4. Deployment Level Hierarchy Description 62
4.4.5. Structure of Dimensions, Groups, and Tenants 63
4.4.6. Description of Deployment Models 65
4.4.7. Completeness of Deployment Constraints 66
4.4.8. Definition of the Deployment Information 67
4.5. Prototypical Realization 68
4.5.1. Description of Realization Approach 69
4.5.2. Implementation of the Model using OWL 70
4.5.3. Model checking 73
4.5.4. Extraction of Deployment Information 75

4.6. Evaluation **77**
4.6.1. Example Environment 78
4.6.2. Scenario Definition 80
4.6.3. Analysis of Results 81
4.7. Summary **84**

Make everything as simple as possible,
but not simpler.

Albert Einstein

In Section 2.2.3 the research questions were introduced and motivated that are to be tackled by this work. Research question RQ-1 deals with capturing of Customers' Deployment Constraints towards Mixed-Tenancy. The primary focus of this chapter is to address this particular question and to describe an approach that is capable of solving the problem. In order to do so, the creation of the following artifacts are described in this chapter.

Description Model The first artifact is a formal and complete definition of the Description Model that allows Customers to specify their Deployment Constraints.

Definition 16 (Description Model) The *Description Model* is an application independent model that allows to capture Deployment Constraints towards a Mixed-Tenancy deployment. It is formally and completely defined by this work.

The primary objective of the Description Model is to be as generic as possible. This means that the Description Model is supposed to be reusable for a wide variety of different Multi-Tenancy applications and environments. Thus, the Description Model is the framework that defines which Deployment Constraints may actually be expressed. The Description Model directly contributes to research question RQ-1.3.

Utilization Process The Utilization Process specifies the steps that are necessary to apply the Description Model to a specific application. This is due to the fact that the Description Model is not specific to any particular application, but is designed to be generic. As a result, the defined steps are not only performed by the Customers, but also by the Operator of the application.

Definition 17 (Utilization Process) The *Utilization Process* defines the necessary steps to apply the Description Model to a specific application.

The Utilization Process directly contributes to research question RQ-1.2.

Prototypical Realization The final artifact described by this chapter is a prototypical realization of the Description Model. Based on it, it is possible to execute the steps defined by the Utilization Process. This realization is implemented using technologies from the area of Semantic Web since they provide beneficial means for realization. The realization represents one integral part of the Deployment Configuration Generator (introduced in Section 2.2.5). Further, the results will be applied within the case study in Chapter 6. Thus, it will contribute to research question RQ-3.

In order to create these artifacts, the chapter is structured as follows. The chapter starts with an introduction of fundamental concepts that are relevant for the understanding of this chapter (Section 4.1). This section's purpose is to support readers without knowledge of first-order logic and graph theory. The actual content of this chapter starts with Section 4.2 by conducting a conceptual analysis of the requirements which Operators and Customers may have towards the description of Mixed-Tenancy Deployment Constraints (contributing to research question RQ-1.1). By doing so this section lays the foundation of the Description Model and, thereby, determines what is going to be describable by the Description Model. Section 4.3 introduces the Utilization Process of how the Description Model may be utilized by an Operator. The overall process as well as the individual steps are discussed in this section on a high level to introduce the general idea. In Section 4.4 the Description Model is introduced that allows to capture Customers' Deployment Constraints. This is done in a complete, formal, and technologies-independent way by applying first-order logic. Based on the formal description, the chapter continues by discussing one possible way of realization. This is done by Section 4.5. Based on the realization, Section 4.6 continues by evaluating the approach. This is done by discussing an example that captures a variety of possible cases. The purpose of this is to demonstrate that the example realization, that was created based on the formal Description Model, provides the capabilities that were analyzed in the beginning of this chapter. The chapter concludes with Section 4.7 by summarizing the results of this chapter.
Previously, this chapter has partially been published in [RWV13].

4.1 Fundamental Concepts and Approaches II

This section introduces the fundamental concepts and approaches that are relevant for this chapter. For this chapter first-order logic will be used to formulate the Description Model. Thus, at this point a brief introduction is given and the notation used in this work is defined. Furthermore, within the Description Model it is necessary to define graphs for means of representation. Thus, this section will give a simple introduction of graph theory and standard graphs relevant to this work.

4.1.1 First-order Logic

The goal of logic in computer science is to serve as a language that allows to create models that represent situations encountered by computer science professionals. Using these models it is possible to reason formally and, based on that, conduct new arguments. The ability to do that formally, bears the advantage that arguments are valid and can be defended rigorously [Hut11].

Propositional logic is a formal system in which formulas of a formal language may be interpreted to represent propositions (e.g. "if company A is using the application, company A is a Customer). In addition to propositional logic, first-order logic is a formal system that allows to make general propositions about entities. This is achieved by introducing the universal quantifier (read: "for all") and existential quantifier (read: "there exists" or "for some") [Hut11]. The universal quantifier allows to express universal statements about certain elements (e.g. "all companies that use the offered application are Customers"). The existential quantifier allows to express universal statements about the existence of elements (e.g. "all Customers must have a User").

The notation used within this work to express logical statements is defined by Table 4.1.

4.1.2 Introduction of Graph Theory

In mathematics and computer science graphs serve as mathematical models that allow to analyze many concrete problems in the real-world. A graph in this context is an abstract representation of a set of objects where some pairs of these objects may be connected. The objects are represented by abstractions called vertices, and connections between vertices are called edges [Wal07; BR12]. A graph is commonly defined as a pair $G = (V, E)$, with a finite set of vertices V and a set of edges $E \subseteq V \times V$. A graph is visualized through diagrams by depicting dots representing the vertices and lines or curves representing edges. Graphs may be used to represent a wide variety of scenarios in the real-world and to formulate concrete problems.

For this work the following terms shall be used.

Adjacent Two vertices are adjacent if they are connected by an edge.

Loop A loop is an edge connecting a vertex with itself.

Walk A walk in a graph is a finite sequence of vertices in which two succeeding vertices are connected by a node.

Cycle A cycle is a walk through a graph in which start and end are the same vertex.

Undirected Edge An undirected edge is one which has no direction. Thus, it is an unordered pair of vertices that can be represented graphically as a line between two vertices.

Symbol	Example	Interpretation
$\{\ \}$	$\{x_0,\dots,x_9\}$	Set containing $x_0,\dots,x_9$
$\mid$	$\{x \mid T(x)\}$	Set containing all x that satisfy the condition $T(x)$
$\cup$	$A \cup B$	Union of set A and B
$\cap$	$A \cap B$	Intersecting of set A and B
$\times$	$A \times B$	Cartesian product of set A and B
$\setminus$	$A \setminus B$	Difference of A and B (elements in A, but not in B)
$\subset$	$A \subset B$	A is a subset of B
$\subseteq$	$A \subseteq B$	A is a subset of (or is included in) B
$\sqcup$	$A \sqcup B$	the sets A and B are disjoint
$\in$	$a \in A$	Element a is part of set A
$\notin$	$a \notin A$	Element a is not part of set A
$\lvert\ \rvert$	$\lvert A\rvert$	Cardinality of set A
$\wedge$	$A \wedge B$	Statement A and statement B
$\vee$	$A \vee B$	Statement A or statement B
$\oplus$	$A \oplus B$	Either statement A or statement B
$\rightarrow$	$A \rightarrow B$	Statement A implies statement B
$\leftrightarrow$	$A \leftrightarrow B$	Statement A is equivalent to statement B
$\forall$	$\forall x$	For all elements x
$\exists$	$\exists x$	There exists at least one element x
$\exists!$	$\exists! x$	There exists exactly one element x
$\nexists$	$\nexists x$	There exists no element x
$:$	$\forall x \colon T(x)$	For all x that satisfy the condition $T(x)$ is true
$\sqsubseteq$	$\mathcal{C} \sqsubseteq \mathcal{C}'$	$\mathcal{C}$ is a refinement of $\mathcal{C}'$ (defined in Section 5.1.3)

Table 4.1.: Introduction of used Notation

Directed Edge A directed edge is an edge having a direction associated. Thus, it is an ordered pair of vertices that can be represented graphically as an arrow drawn between two vertices.

Graph theory distinguishes many different types of graphs. Those, relevant for this chapter, are introduced in the following.

Undirected Graph An undirected graph is a graph with undirected edges that may have loops and cycles.

Simple Graph A simple graph is an undirected graph that contains no loops.

Directed Graph A directed graph is a graph with directed edges that may have loops and cycles.

Directed Acyclic Graph A directed acyclic graph is a directed graph without cycles. This means there is no way to start at some vertex and follow a sequence of edges that eventually loops back to the start.

Complete Graph A complete graph is a graph in which every vertex is adjacent to every other vertex.

4.2 Conceptual Analysis based on Requirements

This section marks the first step towards the definition of the Description Model. First, it does that by analyzing the requirements about what the Description Model it is supposed to be able to capture. Thereby it is determining the expressiveness of the Description Model. Second, for every requirement a conceptual design is proposed[1].

The requirements were identified by an analysis of the problem domain. Furthermore, interviews and discussions were conducted with experts from both industry and academia. The interviewed industry experts were all employed in globally operating IT companies, most worked in technical roles with contact to Customers. Thus, they had expertise about the market demand as well as good understanding about realization. The experts from academia were all doing on edge research in related fields. Some of them were met at conferences, for others specific meetings were arranged. Based on these discussions and the gathered feedback, the model was refined several times. Furthermore, aspects that came up in these interviews that were not included in the work, may be found in the future research section (Section 7.1.2). However, since expert interviews may not be conclusive, Chapter 6 will analyze the applicability of the Description Model to real-world scenarios in detail by performing a case study. This will also give conclusions about the quality of the requirements that were introduced by this section.

The section starts by analyzing the environments in which Multi-Tenancy applications shall be deployed and, based on this, determines requirements to be met (Section 4.2.1). This is followed by an introduction of the Deployment Models that shall be supported (Section 4.2.2). Deployment Models refer to different styles of how Customers may express their Deployment Constraints. Some of these Deployment Models allow to exclude or include individual Tenants or entire Groups of Tenants. Thus, the grouping of Tenants is discussed in Section 4.2.3 and the creation of collections of Groups, so-called Dimensions, is discussed in Section 4.2.4.

It is a goal of the approach to keep Customer base secret. To be able to achieve this goal, it is necessary to tackle a challenge that is discussed in Section 4.2.5.

4.2.1 Levels of Deployment

In order to create a Description Model that shall be able to capture Customers' Constraints towards Mixed-Tenancy deployments, it is necessary to take a look to what a deployment has to look like. The goal of this work is to deploy composite Multi-Tenancy applications following the Mixed-Tenancy paradigm, where every Tenant may give Constraints to if or with which other Tenants they are willing to share resources.

As discussed in Section 2.1.3, Multi-Tenancy is a paradigm that allows multiple Tenants to share the same instance of an application and the entire underlying infrastructure stack. A Single-Tenancy application, on the other hand, is not shared

[1] A complete formal definition of all requirements is given by Section 4.4.

by multiple Tenants. It may, however, be possible that the Operator chooses to have Tenants share some of the infrastructure stack. An example may be that the Customers each use their own instance of the application (Single-Tenancy), but all those instances are being deployed in the same virtual infrastructure.

It is the goal of this work to allow Customers to express their Deployment Constraints for different layers of the infrastructure stack to increase their security by still allowing the Operator to make use of economies of scale. Those layers for which Deployment Constraints shall be expressible, this work refers to as Deployment Levels.

Definition 18 (Deployment Level) A *Deployment Level* (or only Level) is a slice of the infrastructure stack for which the Operator offers the possibility to realize Mixed-Tenancy. A slice encapsulates one to many layers of the infrastructure stack[2]. Each Deployment Level needs to apply to the rules that will be discussed in the following.

Definition 19 (Deployment Unit) There will be one or many instances of every Deployment Level, once a deployment has been created. These instances are referred to as *Deployment Units*.

Equal to the infrastructure stack, the Deployment Levels of one Application Component are structured as stacks, each having none or one predecessor (lower) and none or one successor (higher) Deployment Level. Whether an infrastructure layer can/shall become a Deployment Level depends on if the following rules apply:

A Deployment Level for Instances of Application Components needs to be Defined This is due to the fact that this work is dedicated to composite Multi-Tenancy applications. In order to realize the proposed Mixed-Tenancy approach, it is necessary that all Application Components are associated to this Level. This Level is the highest Deployment Level of the stack of all Application Components.

It is possible to create multiple Units of a Deployment Level This is a necessary requirement since for this work separating Tenants means having them use different instances. If this requirement could not be met, the entire approach would not work.

The next (lower) Deployment Level shall be able to run multiple instances of the current Deployment Level simultaneously This requirement is necessary, as otherwise it would not be possible for the Tenants to express their Constraints for every Deployment Level independently. An example for a Deployment Level where this requirement cannot be met is *operating system* over a *virtual machine*. Given the case that a Tenant expresses the requirement to use an individual instance of an operating system, this would result in an individual instance of a virtual machine. This is due to

[2]The concept of the infrastructure stack was introduced in Section 2.1.3.

the fact that a virtual machine will not be able to run multiple operating systems simultaneously. In case the Deployment Levels *application server* and *virtual machine* were created, it would be possible that a Customer with the requirements of not sharing the same application server requirement may still share the same virtual machine with other Tenants. Thus, this requirement contributes to the Operator's objective of using infrastructure as efficiently as possible, but still it allows the Tenants more freedom of defining their Deployment Constraints. This objective has previously been introduced by Section 2.

Sharing of data between different Units of the same Deployment Level shall be establishable if necessary In order to have a working platform, it may be necessary that the instances of a particular Deployment Level may share data. An easy example for this is creating a Deployment Level for Application Components. In this case all instances that are used by the same Tenant will need to share data. A more challenging example is an application server. One of the tasks of an application server is to hold the session that stores data related to the User currently using the application. Thus, the application server Units that belong to the same Tenant (and thereby to the User) will need to share the session data. For the proposed system this is not a trivial task.

However, there may be Deployment Levels where communication may not be necessary (e.g. virtual machine). In this case this requirement does not need to be considered.

As this work deals with composite applications, it is possible that not all Application Components share the same Deployment Level stack. It may be the case that different Application Components rely on different Infrastructure Stacks and, therefore, on different Deployment Level stacks. In fact, it would be possible that two Application Components belonging to the same application do not share Deployment Levels at all, except of course the first one (which was stated to be mandatory) that states that both are Application Components. Thus, it is necessary that the Description Model has the ability to capture multiple Deployment Level stacks.

It may also be possible that the same Application Components of the same composite application rely on similar but slightly different stacks. As an example, take a scenario where all Application Components are being deployed on virtual machines, but some also require an application server and some do not. In this scenario both types of Application Components may share the same virtual machines if Customer Constraints permit it. In order to be able to realize this, it will be necessary to capture all stacks of Deployment Levels. Since all Application Components must belong to the first Deployment Level called *instance*, all Deployment Levels relevant for the deployment of an application can be captured as a directed acyclic graph. The Application Components need to be associated to those Deployment Levels they shall be deployed on (which must always be a stack). An example that captures all possible stacks that shall be supported by the Description Model is illustrated in Section 4.6.1.

4.2.2 Deployment Models

For every Application Component and Deployment Level the Customers have to choose the Deployment Model according to which they want to express their Deployment Constraint.

Definition 20 (Deployment Model) A *Deployment Model* is a blueprint of a Deployment Constraint. It defines the characteristics of if or how Tenants will be able to express their Deployment Constraints for their deployment or a particular Application Component and Deployment Level.

As discussed in Section 4.2.1, it is necessary for Customers to make that choice multiple times since Deployment Constraints need to be defined for every Deployment Level and every Application Component. The same Deployment Model may be applied to multiple (or even all) Application Components and/or multiple (or even all) Deployment Levels. This means it is possible to specify that the entire application is shared freely and publicly with other Tenants, on all Deployment Levels. Application Components that process very sensitive data may be assigned to more restrictive Deployment Models.

The following are the possible Deployment Models that shall be supported by the Description Model. Each Deployment Model may be applied on every Deployment Level.

Private

Following this Deployment Model, a Deployment Unit is deployed specifically for a single Tenant. This specific Unit is not shared by multiple Tenants. If applied to a specific Application Component on the instance Deployment Level, the entire application stops being a pure Multi-Tenancy application since this deployed Application Component does not need to support Multi-Tenancy anymore. In fact, it would be possible that a Customer demands that all Application Components are deployed following the private model. This would lead to a Single-Tenancy deployment of a Mixed-Tenancy application. If another Deployment Model would be chosen for lower Deployment Levels, it would still be possible for the Operator to do some resource usage optimization.

Public

If this Deployment Model is chosen, the Tenant states that sharing is permitted with all other Tenants. Thus, Tenants using this Deployment Model have no influence on with whom they are deployed. As this is the easiest way for the Operator to deploy the application, it should also be the cheapest for the Customer. For Customers, this may also be the right way for Application Components that do not handle sensitive data.

White Hybrid

This Deployment Model allows Customers to specify the choice with which specific Tenants or Groups of Tenants they feel comfortable to share Deployment Units. A real-world application of this might be that a company demands that individual departments (belonging to the same company group) share an instance, but any external Tenants are not allowed to use the same instance. Furthermore, this Deployment Model allows to manage collaboration if applied to the instance Deployment Level. It can be used to specify which specific Customers want to share the same Application Component instance that allows collaboration, for example, by deactivating the isolation of a Mixed-Tenancy component. However, this is not further elaborated in this work.

Black Hybrid

In this Deployment Model Customers can specify with which other Tenants or Groups of Tenants they do not want to be deployed. A real Customer demand that can be fulfilled using this Deployment Model is that a Customer demands not to be deployed with competitors (without specifically specifying who they are) or they demand not to be deployed with another specific Customer.

Gray Hybrid

This Deployment Model allows Customers to do both, specifying with whom they want to be deployed and with whom they do not wish to be deployed. Thus, it provides the highest level of liberty to Customers to describe their Deployment Constraints. The Deployment Model allows Customers to specify, for example, that they want to share a Unit only with Tenants from a specific geographic region (e.g. Asia) but not with Tenants from a particular industry (e.g. the ones they operate in).

Deployment Model Overview

In the previous paragraphs the *Deployment Models* Private, Public, White Hybrid, Black Hybrid, and Gray Hybrid were introduced. These are the only Deployment Models that shall be supported by the approach of this work. Each Deployment Model has different capabilities to express the Constraints Tenants need to meet in order to be deployable with the Tenant that expressed them. Thus, the way deployable Tenants are determined is different depending on the chosen Deployment Model. This may simply be expressed using set theory. Figure 4.1 illustrates this using Venn-Diagrams. A Gray Hybrid deployment, for example, may only deploy those Tenants that were included and not excluded.

As visible in the figure the Gray Hybrid model would be sufficient to model White Hybrid and Black Hybrid Constraints. This work chose to introduce all five Deployment Models since it will very easily be possible to associate different pricing strategies to each Deployment Model. This, however, is not further elaborated in this work.

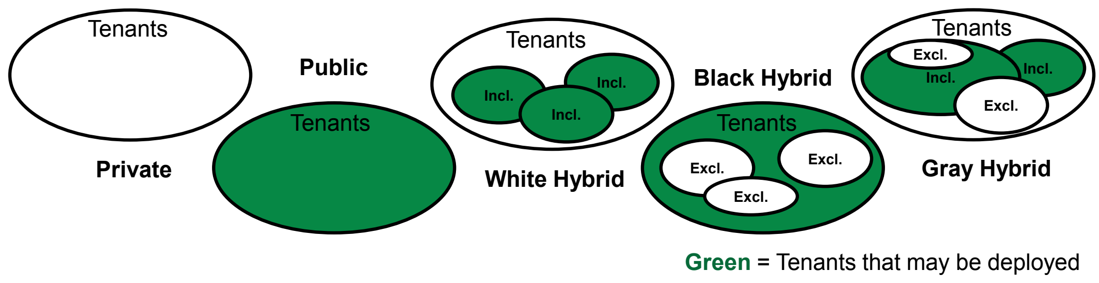

Figure 4.1.: Definition of Deployment Models using Venn Diagrams

4.2.3 Groups

In the previous Sub-Section five Deployment Models were introduced. Some of them provide the ability to the Customer to explicitly specify with which other Tenants they wish or do not wish to share infrastructure. This inclusion and exclusion, however, shall not only be done based on individual Tenants. Furthermore, it shall be possible to exclude or include entire Groups of Tenants.

Definition 21 (Group) A *Group* is an entity that represents a commonality multiple Tenants may share. All Tenants that share this commonality are associated to it.

For example, it would be possible to create a Group that represents a particular industry (e.g. IT, TC, Pharmaceutical). All Tenants that belong to this industry would be associated to it. Once this is done, it would be possible to easily realize the requirement that a Tenant wishes not to share infrastructure with competitors. It would only be necessary to exclude the particular industries the Tenant operates in. Thus, all Tenants associated to this Group (which means operating in those industries) would be excluded.

4.2.4 Dimensions

In the previous Section the requirement has been discussed that it shall be possible to categorize Tenants by grouping them. The Section gave an example for this by grouping Tenants according to industries they operate in. This industry-based grouping is only one possible way of grouping Tenants. There are a lot of others possible. Within this work, the ways of how Tenants shall be grouped are refered to as Dimensions.

Definition 22 (Dimension) A *Dimension* is an entity that represents one possible subject according to which Tenants may be grouped. All Groups that contribute to this Dimension are associated to it. Thus, as *Groups* represent commonalities Tenants may share, a Dimension is in fact a collection of commonalities that all contribute to the same subject.

The definition of one to many Dimensions shall be possible within the same scenario. Thus, the Description Model proposed by this work is not limited to any particular Dimension. In fact, the idea is to have the Operator decide which and how many Dimensions they need for a specific application and target market. Each Dimension is realized by a collection of Groups. It shall be possible to organize the Groups realizing the same Dimension as a directed acyclic graph (in which the root is the Dimension). Figure 4.2 illustrates the proposed structure to realize Dimensions. As visualized by the figure it is possible that Groups have

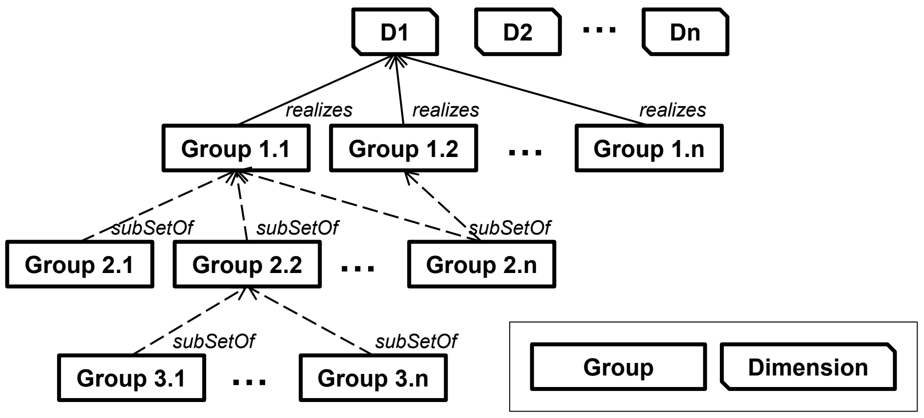

Figure 4.2.: Description of Dimensions and Group Structure

Subgroups that are a sub-set of them. The capability of structuring Groups like that allows a much finer-grained categorization of Tenants. An example would be a Tenant being part of a certain sub-industry that belongs to a particular industry (e.g. banks or insurances within the finance industry). However, once Customers express to include or exclude a particular Group, they will also include or exclude all Groups that are Subgroups of them (including Groups that belong transitively to the Groups). This means excluding the finance industry will result in the exclusion of banks and insurances.

4.2.5 Virtual Tenants

Previously, it was stated that Tenants shall be able to explicitly include or exclude other Tenants (in addition to Groups). This may lead to a problem as a Tenant might want to exclude companies that have not been Tenants of the application yet.

The idea of this work is to tackle this problem with the goal of keeping the Customer base secret. This may be accomplished by introducing the concept of Virtual Tenants.

Definition 23 (Virtual Tenant) A *Virtual Tenant* is a company that exists in the market or might potentially exist but is not yet a Tenant of the application.

In the description of Deployment Constraints, the Virtual Tenants are treated similarly to regular Tenants. The only difference is that only for regular Tenants the deployments of the application will be computed and realized.

Once a company becomes a Tenant of the application, the Virtual Tenant is transformed to a regular Tenant. This is done by keeping all previously specified Deployment Constraints valid. This is also true for the opposite case. If a company stops being a Customer, but there are still Constraints to consider, the Tenant is transformed into a Virtual Tenant.

4.3 Process of Deployment Constraint definition

The previous section performed a first conceptual analysis of the requirements Tenants may have towards the description of Mixed-Tenancy deployments. Based on this, this section presents the Utilization Process as a next step towards the solution of Research Question RQ-1.2.

Due to the fact that the solution of this work is intended to be as generic as possible, the definition of Deployment Constraints is performed not just by the Tenants but also by the Operator. The Utilization Process defines the necessary steps to capture Customers' Deployment Constraints. This is done by introducing and discussing the necessary steps that belong to the process. The process is illustrated by Figure 4.3

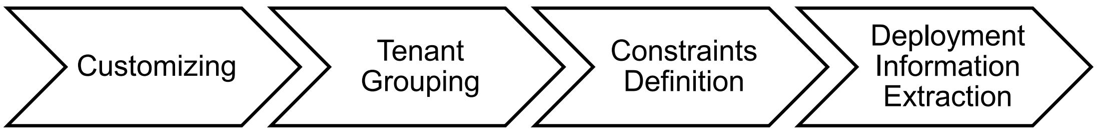

Figure 4.3.: Description of the Utilization Process

The foundation of the Utilization Process is the Description Model that allows to realize the entire description.

4.3.1 Customizing

The first step that needs to be taken in order to apply the approach of this work is called customization. It was an objective of this work to create the Description Model the way that it is as generic as possible, in order to be applicable to a variety of applications. This is why this first step of the application process deals with the customization of the Description Model to the specific application that is supposed to be offered. Thus, this step is performed by the Operator.

The customization step consists of three activities that need to be performed. They will be discussed in the following.

Entity Capturing

As stated before, it is the goal of this work to propose a generic model. Thus, the first step is to capture the important entities for the given scenario. This includes the Tenants that wish to use the application and the Application Components. This is the purpose of this first customization activity.

Deployment Level Definition

This activity's purpose is the definition of the Deployment Levels. Section 4.2.1 introduced Deployment Levels and stated the rules all Deployment Levels need to apply to. When the Operator defines the Deployment Level for their application, these rules need to be fulfilled. Furthermore, it was stated that Tenants will have the ability to express their Deployment Constraints for some of the Deployment Levels.

Since the decision whether or not Deployment Constraints may be expressed for a Deployment Level has impact on how optimal available resources may be utilized, Operators should allow the definition of Constraints only for those Deployment Levels where Customers demand it. However, there is no restriction to the number of Deployment Levels that may be created.

Dimension and Group Definition

Dimensions are subjects according to which Tenants need to be categorized. Based on this categorization, Tenants may exclude or include other Tenants from sharing resources. This was introduced and discussed in Section 4.2.4.

The goal of this activity is the definition of the Dimensions and the Groups that realize them. For this, the Description Model only serves as the framework. It does not give a restriction to the number of Dimensions or the number of Groups realizing each Dimension. It is up to the Operator to determine the specific Dimensions they want to provide to their Customers. This gives the Operator the chance to take the requirements of their Customers into account and define Dimensions according to their needs. Once a Dimension has been defined, the creation of Groups needs to be done based on the target Customer base. Since Groups will represent the commonalities that Customers have, the target Customer base will give some insight on which Groups will be necessary. For example, if the Dimension "Industries" was created, the industries in which the target Customers operate, need to be captured by the Groups. The level of abstraction Groups represent, however, needs to be determined by the Operator. For the "Industry" example, it would be possible to create a specific Group for banking and insurance, or a generic one for finance. Of course it would also be possible to create the Groups banking and insurance that are Subgroups of a finance Group.

The way the Operator defines the Groups has an impact on how well resources may be utilized. This is due to the fact that the more precise Customers may express their Deployment Constraints, the less restrictive will their Constraints be in the end. An example for this is a Customer that does not want to be deployed with banks. If there is such a Group, the exclusion is limited to Tenants that are banks. If the Operator only created a finance Group, the Customer will exclude a higher number of Tenants, which leaves the Operator with less space for optimization. Thus, it is in the interest of the Operator to have very detailed means of categorizing Tenants within Dimensions in order to have more space for optimization.

4.3.2 Tenant Grouping

The second step of the process is called Tenant grouping. Similar to the previous step it is also executed by the Operator. Its goal is to categorize all Tenants that wish to use the application. This means that it is necessary to execute this step every time a new Tenant is introduced to the system or the data of an existing one changes. Each Tenant needs to be assigned to at least one Group that realizes a particular Dimension (this may be done transitively).

The reason that this needs to be done is that otherwise it would not be certain that excluding a particular Group may actually result in excluding the Tenants that apply to this commonality. Even though this process step seems easy and straight forward at first it is not, once looked at it closer. The Tenant grouping is a crucial step in order to create data on which Constraints may be defined. If this data is flawed, Deployment Constraints defined by Tenants may not be properly realized. For example, it may be possible that an Asian company is not associated to the Asian Group (not directly nor transitively). If this is the case, an exclusion of the Asian Group will not lead to a Deployment Constraint without this particular Customer.

Problems caused by flawed data may appear due to a wide variety of reasons. For example, a company may extend or limit their area of business (e.g. regionally or industry wise), or two companies may merge. Furthermore, it might be possible that a company specifically tries to flaw data in order to be deployed with a competitor.

In order to avoid such things from happening, it is crucial that validation processes and techniques are established that guarantee data validity. However, this work is limited to the techniques of how these requirements may be described. Processes and techniques to keep data quality accurate are out of scope of this work. However, a first idea of how this problem may be tackled is briefly discussed in Section 7.1.2.

4.3.3 Constraint Definition

In the previous Subsection it was explained how Dimensions are defined and Tenants are grouped. Furthermore, in Section 4.2.2 the Deployment Models were introduced that are supposed to be offered to Customers. This Section now puts all that together and discusses how Customers may express their Deployment Constraints of a particular Application Component and Deployment Level.

Figure 4.4 gives an example. In order to describe the Deployment Constraints, the Customer has to start by picking a Deployment Model. Then, the Tenant has to specify to which Application Component (or Application Components) and Deployment Level (or Deployment Levels) the Deployment Constraint shall apply to. Once this is done, the final step is to give the Deployment Model specific constraints. For a public or private deployment there is nothing to do since the selection of the Deployment Model already specifies completely how infrastructure may be shared. For other Deployment Models, however, it is necessary to give more information. For a Gray Hybrid, for example, it is necessary

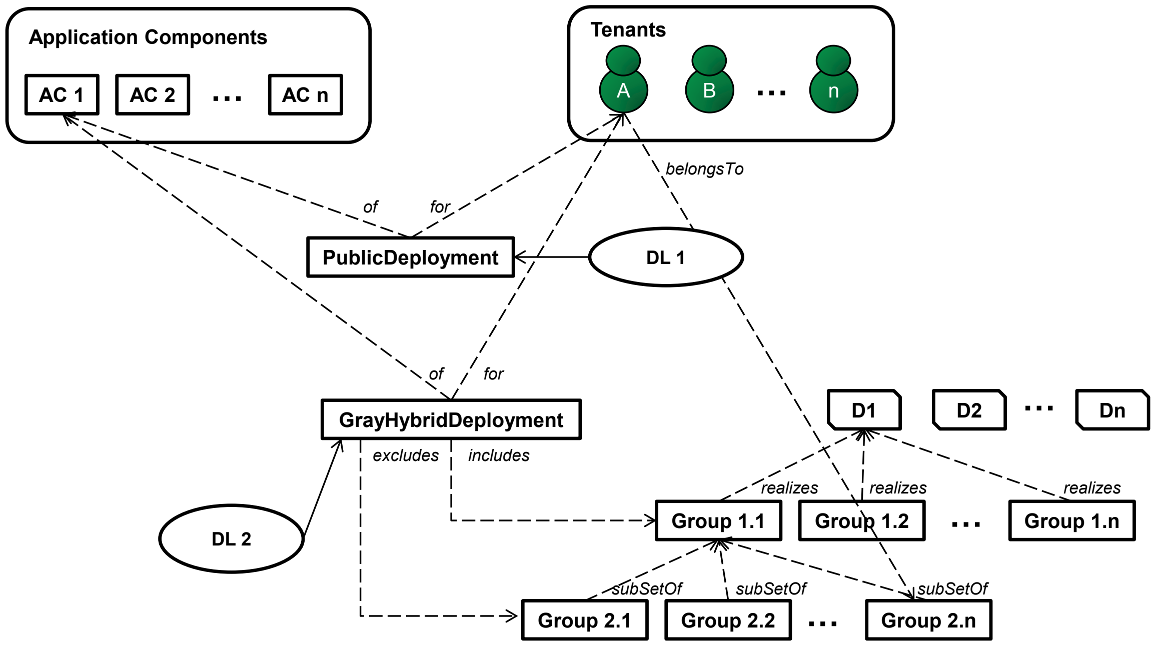

Figure 4.4.: Description of the Deployment Constraint

to explicitly specify which Groups or Tenants shall be excluded and which shall be included. Figure 4.4 indicates this by a deployment called "GrayHybridDeployment", which is for Tenant A and deploys Application Component 1. In this example Tenant A specifies that they want to share an instance only with Tenants associated to Group 1.1. Furthermore, they express that they do not want to share this instance with Tenants associated to Group 2.1. According to the specification of the Gray Hybrid model, this means that Tenant A only wants to share an instance with Tenants associated to the Groups 1.1, 2.2, . . . , 2.n.

As discussed in Section 4.2.1 it is necessary that a Tenant specifies the Deployment Constraint of every Application Component multiple times, once per Deployment Level. This is why in the example of Figure 4.4 Tenant A specifies two Deployment Constraints of Application Component 1, one which is on Deployment Level 1 and one which is on Deployment Level 2. Furthermore, due to the fact that the Deployment Units will be cascading into each other (according to the rules defined in Section 4.2.1), it is necessary that lower-level Deployment Constraints are equally or less restrictive than their predecessors. This will be further elaborated in Section 5.

Since this step is performed by the Customer, it would be necessary to create an easy-to-use user interface, for example, as part of a self-service portal. However, this is out of scope for this work and is listed for future research (Section 7.1.2).

4.3.4 Deployment Information Extraction

Once all Tenants have expressed their Deployment Constraints for all Application Components they use on all relevant Deployment Levels, it is possible to extract the information that is needed for calculating Valid and Optimal Deployments.

The information needed to do so is whether two Tenants are allowed to share infrastructure or not. This needs to be done for every Application Component, once per relevant Deployment Level. The information may be gained by the

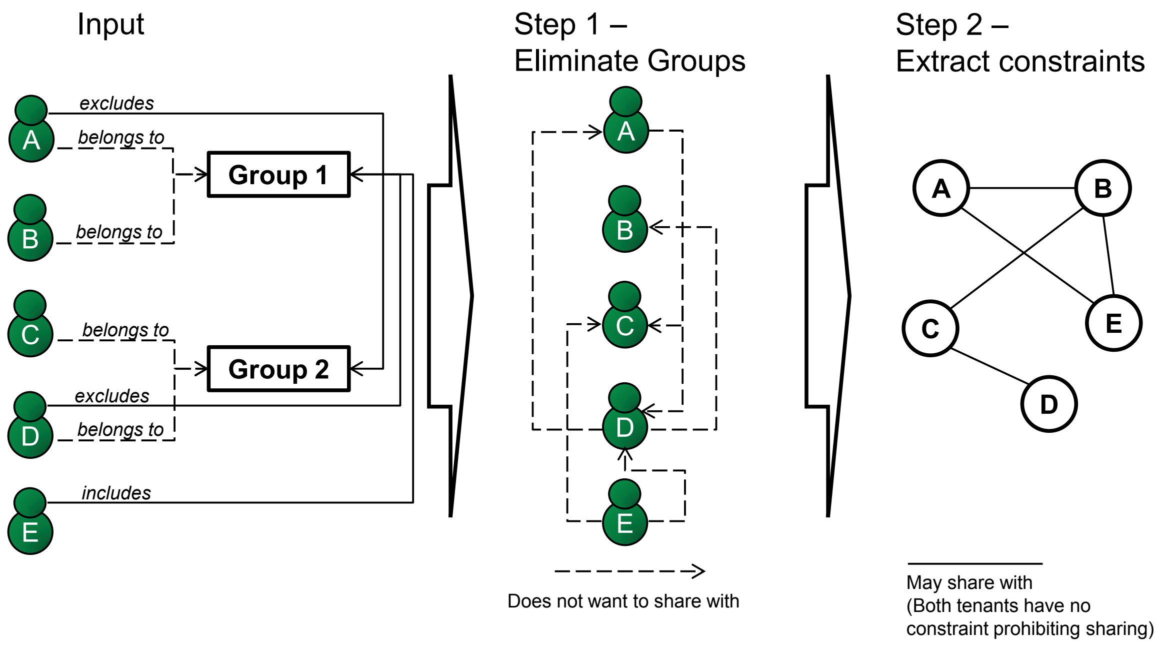
Input
Step 1 – Eliminate Groups
Step 2 – Extract constraints
excludes
belongs to
Group 1
Group 2
includes
A
B
C
D
E
Does not want to share with
May share with (Both tenants have no constraint prohibiting sharing)

Figure 4.5.: Description Deployment Information Extraction

activities that are illustrated by Figure 4.5. The left column represents an example input where five Tenants are grouped into two Groups. These Groups are included and excluded by different Tenants in order to define their Constraints. Please note that as Deployment Information extraction is done for every Application Component and relevant Level, only those Groups are part of the input that have an impact on the Deployment Constraints.

Based on this input, unnecessary information may be eliminated. Thus, step one (Figure 4.5 middle) starts by removing the Groups from the input. Instead of including or excluding Groups, Tenants now directly exclude other Tenants. The include relationship of Tenant E was also transformed into exclude relationships.

Based on this, it is possible to extract which Tenants may share the resource by checking if each combination of Tenants has Constraints between them or not. This information can easily be represented by an undirected graph. For this graph the nodes represent Tenants and edges represent whether two Tenants may share infrastructure or not. In case two Tenants may share resources, they would be connected by an edge. This is only the case if neither one of them has Deployment Constraints that express that they shall not share resources.

Such a graph is needed for every Application Component, once per relevant Deployment Level. All those graphs shall automatically be created once all Deployment Constraints have been described by the Tenants since they represent an input for the Deployment Computation Algorithm. A prototypical implementation is described by Section 4.5 and an example application of this work is described by Section 4.6.

4.4 Generic Mixed-Tenancy Description Model

Section 4.2 dealt with a requirement analysis of what the Description Model, that is to be created in this chapter, shall be able to capture. Further, it introduced a first conceptual analysis for the Description Model. Based on this, Section 4.3 discussed the process of how the Description Model may be applied to a wide variety of applications. Based on these two Sections, it is this section's purpose to introduce the Description Model that is capable of realizing all previously discussed requirements. This shall be done by providing a complete, formal, and technology-independent model. To achieve this goal, first-order logic will be utilized.

In order to do so, this section starts with a first, incomplete description of the Description Model using the class diagram notation of the Unified Modeling Language (UML). The purpose of this is to give a point of reference to the reader. This allows to grasp the basic idea of the model quickly. Based on this incomplete discussion, the section continues by defining all aspects of the Description Model in detail.

4.4.1 Introduction of the Fundamental Idea

As stated by the introduction, this Section starts by giving a quick but incomplete overview of the Description Model using UML class diagram notation [PP05; OMG13b]. This is done by Figure 4.6. It illustrates the entities that were discussed

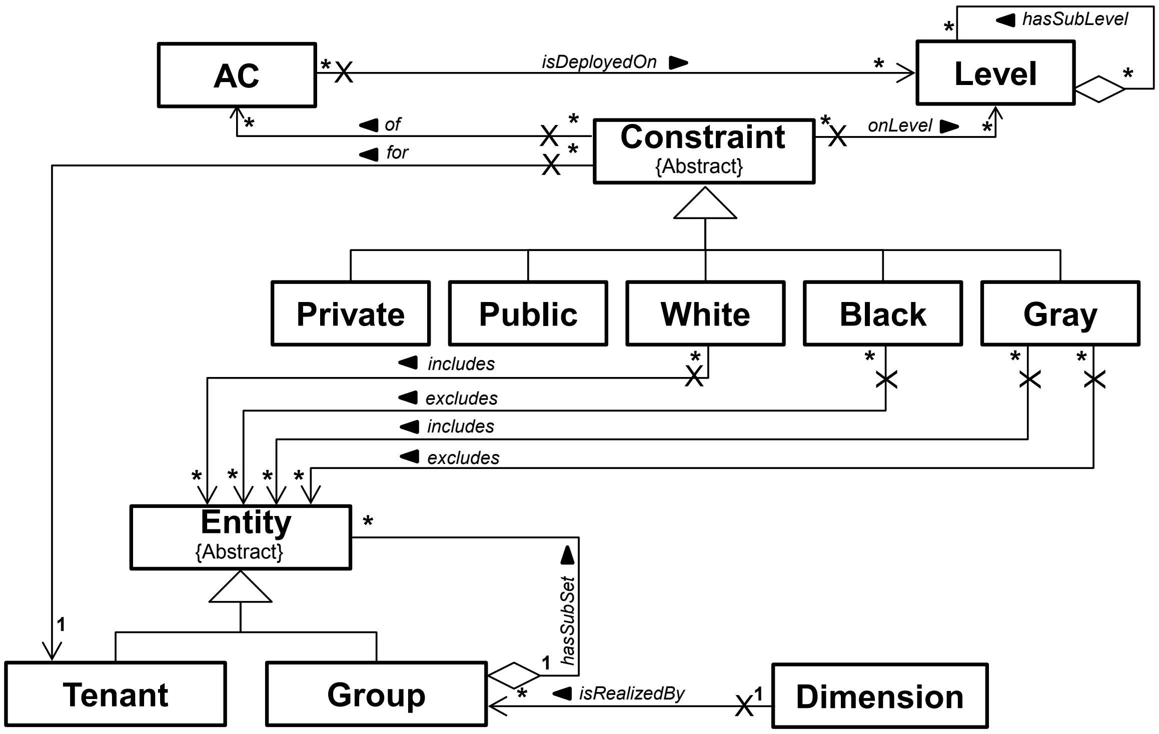

Figure 4.6.: Description Model as UML Class Diagram (incomplete)

in previous sections as classes. Furthermore, there are relations between these classes that may be used to describe Deployment Constraints. In Section 4.3.3 an example was given in which Tenant A defined that they want to use Application Component AC-1 as a public deployment on Deployment Level DL-1 (illustrated by Figure 4.4). Realizing this based on the UML Model would mean that there is an object of the type Application Component, called *AC-1*. In addition, there is an object *A*, of type Tenant and an object *DL-1* of type Level. According to the process defined in Section 4.3, these objects would have been created by the Operator as part of the first step called customizing (Section 4.3.1). Once this setup is made, the Tenant creates the aforementioned Deployment Constraint. In order to do so, they would create an object of type Public (which is of type Constraint). This object is then associated to *AC-1* by using the *of*-relationship, to *A* by using the *for*-relationship and to *DL-1* by using the *onLevel*-relationship. Thus, the constraint is captured. A more complex Constraint could be using one of the other Deployment Models by creating an object of those classes. Depending on the incomplete description, the object would have include or exclude-relationships to Groups or Tenants.

Please note that the structure between Tenants, Groups and Dimensions, as introduced by Subsection 4.2.3 and 4.2.4, is realized in the Description Model using the composite design pattern [GHJ94]. It allows that every Group may have multiple Subgroups as well as that Tenants may be associated to any Group. An inclusion or exclusion of a Group shall also include all Tenants or Groups that are directly or transitively connected to the Group through the *hasSubSet*-relationship. However, this cannot be covered by a plain UML model and, thus, will be defined in the following using first-order logic.

4.4.2 Foundations of the Formal Model

In the previous subsection the basic idea of the Description Model was introduced using UML notation. This is not sufficient in order to have a complete and unambiguous description of the Description Model since there are certain structures that are not possible to be captured by a basic class diagram.

However, it is the goal of this work to provide a complete description of the Description Model. The obvious approach to do that would be to extend the existing superficial UML description with the definition of OCL Constraints [OMG13a]. However, in order to have a more readable description, first-order logic is applied to describe the entire Description Model. Another benefit of using first-order logic for the description is that some of the definitions given in this section may be reused in Chapter 5. This chapter will deal with tackling research question RQ-1.3 and requires some formal definition of the optimization problem.

In order to describe the entire Description Model, the first step is to define sets that represent the entities necessary for description. In addition to the sets representing entities, the relationships between the entities will be defined as finitary relation. The necessary sets and some of the basic functions are defined in the following paragraphs.

Definition of Entities

In order to capture and define all entities that need to be describable in the Description Model, the following defines sets. Each set represents one class of entities and its individuals. They correspond directly to the classes in the UML description (created in the previous subsection).

$$T = \{t_1, \ldots, t_{it}\} \quad : \quad \text{Tenants} \tag{4.1}$$
$$AC = \{ac_1, \ldots, ac_{iac}\} \quad : \quad \text{Application Components} \tag{4.2}$$
$$DL = \{dl_1, \ldots, dl_{idl}\} \quad : \quad \text{Deployment Levels} \tag{4.3}$$
$$D = \{d_1, \ldots, d_{id}\} \quad : \quad \text{Dimensions} \tag{4.4}$$
$$G = \{g_1, \ldots, g_{ig}\} \quad : \quad \text{Groups} \tag{4.5}$$
$$DC = \{dc_1, \ldots, dc_{idc}\} \quad : \quad \text{Deployment Constraints} \tag{4.6}$$
$$PR = \{pr_1, \ldots, pr_{ipr}\} \quad : \quad \text{Private Deployment Model} \tag{4.7}$$
$$PU = \{pu_1, \ldots, pu_{ipu}\} \quad : \quad \text{Public Deployment Model} \tag{4.8}$$
$$WH = \{wh_1, \ldots, wh_{iwh}\} \quad : \quad \text{White Hybrid Deployment Model} \tag{4.9}$$
$$BL = \{bl_1, \ldots, bl_{ibl}\} \quad : \quad \text{Black Hybrid Deployment Model} \tag{4.10}$$
$$GR = \{gr_1, \ldots, gr_{igr}\} \quad : \quad \text{Gray Hybrid Deployment Model} \tag{4.11}$$

of-Relationship

The purpose of the *of*-relationship is to associate a Deployment Constraint to the Application Component this constraint shall apply for. In order to capture this formally, a finitary relation called *of* is defined by definition 4.12. It allows to assign a set of Application Components to a given Deployment Constraint.

Every Deployment Constraint must have at least one Application Component associated. Furthermore, it was stated by Section 4.2.2 that it shall be possible to associate multiple Application Components to the same Deployment Constraint. This allows the Tenants to express that multiple Application Components are deployed in the same fashion. All this is defined by Equation 4.13.

$$of \subseteq DC \times AC \tag{4.12}$$
$$\forall dc \in DC \rightarrow \exists\, ac \in AC \;:\; (dc, ac) \in of \tag{4.13}$$

for-Relationship

The *for*-relationship associates Tenants to the Deployment Constraints they have defined. Thus, Equation 4.14 defines the *for*-relation the way that it relates a Deployment Constraint to a Tenant.

It shall not be possible to have multiple Tenants sharing the same Deployment Constraint. Thus, a Deployment Constraint is always only associated to exactly one Tenant. This is expressed by Equation 4.15.

$$for \subseteq DC \times T \tag{4.14}$$

$$\forall dc \in DC \rightarrow \exists!\, t \in T \;:\; (dc, t) \in for \tag{4.15}$$

onLevel-Relationship

It is also necessary to associate a Deployment Constraint to the Deployment Levels it shall apply for. This is done by the *onLevel*-relation.

Equation 4.16 defines the $onLevel$-relation to relate a set of Deployment Levels to any given Deployment Constraints. Furthermore, every Deployment Constraint shall apply to at least one Deployment Level because otherwise it would not be used. The application of the same Deployment Constraint on multiple Deployment Levels shall be possible as well. This is defined by Equation 4.17.

$$onLevel \subseteq DC \times DL \tag{4.16}$$

$$\forall dc \in DC \rightarrow \exists\, dl \in DL \;:\; (dc, dl) \in onLevel \tag{4.17}$$

4.4.3 Application Components and Deployment Levels

In Section 4.2.1, Deployment Levels were introduced and it was stated that different Application Components may be associated to different stacks of the same Deployment Level-graph. In order to be capable of associating an Application Component to a Deployment Level, a relation is required. Equation 4.18 defines the $isDeployedOn$-relation that allows to express just that. Furthermore, Equation 4.19 states that every Application Component needs to be associated to at least one Deployment Level. In fact, it has previously been defined (Section 4.2.1) that every Application Component is associated to at least the instance Deployment Level – dl_1. Finally, Equation 4.20 defines that every Deployment Level shall be used by at least one Application Component. This ensures that there are no unused Deployment Levels in the model.

$$isDeployedOn \subseteq AC \times DL \tag{4.18}$$

$$\forall ac \in AC \rightarrow (dl_1, ac) \in isDeployedOn \tag{4.19}$$

$$\forall dl \in DL \rightarrow \exists ac \in AC : (dl, ac) \in isDeployedOn \tag{4.20}$$

Based on the definition, it is also possible that an Application Component is deployed on multiple Deployment Levels. The minimum, however, is that it is deployed on Level dl_1.

Based on this definition, it is possible to introduce one additional constraint in order to define the structure of a Deployment Constraint. The following equation defines that a Deployment Constraint may only be associated to those combinations of Application Components and Deployment Levels where at least one Application Component is actually deployed on the Deployment Level.

$$\begin{aligned}\forall dc \in DC\, \forall ac \in AC\, \forall dl \in DL \;:\; & (dc, ac) \in of \;\wedge\; (dc, dl) \in onLevel \\ & \rightarrow (ac, dl) \in isDeployedOn\end{aligned} \tag{4.21}$$

4.4.4 Deployment Level Hierarchy Description

In Section 4.2 it was discussed that Deployment Levels represent slices of the infrastructure stack. In order to be able to express their hierarchy, it is necessary to introduce a relation called *hasSubLevel* (Equation 4.22).

$$hasSubLevel \subseteq DL \times DL \tag{4.22}$$

In order to be able to only permit a stack structure, it is necessary to define a constraint that prohibits cycles. For this it is necessary to define the transitive closure $hasSubLevel^*$ (Equation 4.23 and 4.24). Based on this, Equation 4.25 defines a constraint that prohibits cycles.

$$hasSubLevel^* \subseteq DL \times DL \tag{4.23}$$

$$\begin{aligned}\forall dl, dl' \in DL \;:\; & (dl, dl') \in hasSubLevel^* \\ & \rightarrow (dl, dl') \in hasSubLevel \\ & \quad \vee (\exists dl'' : (dl, dl'') \in hasSubLevel^* \\ & \qquad \wedge (dl'', dl') \in hasSubLevel)\end{aligned} \tag{4.24}$$

$$\forall dl, dl' \in DL : (dl, dl') \in hasSubLevel \rightarrow (dl', dl) \notin hasSubLevel^* \tag{4.25}$$

Based on these definitions, it is possible to define the hierarchy of Deployment Levels in the shape of directed acyclic graphs.

As described in Section 4.2.1 it is possible that there are multiple stacks used by different Application Components. Furthermore, it is possible that one stack may have alternatives in it. Each Application Component needs to be associated to those Deployment Levels it applies to. However, it may only be possible to associate an Application Component to those Deployment Levels that belong to the same stack and may not be alternatives. This may be expressed by the following constraint.

$$\begin{aligned}\forall ac \in AC\, \forall dl, dl' \in DL \;:\; & (ac, dl) \in isDeployedOn \\ & \wedge (ac, dl') \in isDeployedOn \\ & \rightarrow (dl, dl') \in hasSubLevel^* \\ & \quad \vee (dl', dl) \in hasSubLevel^*\end{aligned} \tag{4.26}$$

It has already been defined that every Application Component needs to be deployed on Level dl_1 (Equation 4.19). However, since every Application Component always needs to belong to an entire stack, it has to be defined that every Application Component must also belong to exactly one of the lowest Levels.

This is expressed by the following.

$$\begin{aligned} \forall ac \in AC \; \rightarrow \exists!\, dl \in DL \; &: (ac, dl) \in isDeployedOn \\ &\wedge \; \nexists dl' \in DL : (dl, dl') \in hasSubLevel^* \end{aligned} \tag{4.27}$$

As already discussed in Section 4.2, and already defined by Equation 4.19, Deployment Level dl_1 is the instance Level to which all Application Components need to be associated. However, what has not been defined yet, is that Level dl_1 shall be the only Top Level Deployment Level of the Deployment Level Graph. This is defined by the following.

$$\forall dl \in DL : dl \neq dl_1 \rightarrow (dl_1, dl) \in hasSubLevel^* \tag{4.28}$$

The equation states that every Level is a transitive Sublevel of dl_1. Since the graph does not have cycles, this also states that dl_1 is not a Sublevel of any other Level.

4.4.5 Structure of Dimensions, Groups, and Tenants

In Sections 4.2.3 and 4.2.4 the structure of Groups and Dimensions was discussed. For every Dimension it was stated that there may be multiple Groups realizing this Dimension. Furthermore, it was stated that it shall be possible that Groups may have Subgroups. This Subgroup structure was supposed to be represented in the shape of a directed acyclic graph. The structure was illustrated by Figure 4.2. In Section 4.3.2 it was stated that Tenants shall be assigned to at least one Group per Dimension. In order to be capable of modeling all these requirements, the composite pattern [GHJ94] may serve as a blueprint for a realization that has the benefit that it is easy and straight forward. How this may be achieved is described in the following paragraphs.

Abstraction of Group and Tenant

The first step to describe the desired structure is to create a new set that serves as an abstraction of Tenant and Group. This set is called *entity* (defined by Equation 4.29).

$$E = \{e_1, \ldots, e_{ie}\} \qquad : \quad \text{Entities} \tag{4.29}$$

In order to define entity as an abstraction of Tenant and Group, it is necessary to define an inheritance-relationship between the entities.

$$\forall e : e \in T \oplus e \in G \leftrightarrow e \in E \tag{4.30}$$

As stated above, the Description Model shall only cover the structure of Groups, Tenants and Dimensions. Thus, there shall not be an object of the type entity. This is also expressed by Constraint 4.30[3].

[3]In the UML class diagram (Figure 4.6) this is indicated by defining the class Deployment Constraint as abstract.

hasSubset-Relationship

The next step in order to be able to model the desired structure is the definition of the $hasSubset$-relation that allows to define that a Group may have entities as children. Equation 4.31 defines the relation in this way. This will lead to a directed graph structure.

$$hasSubset \subseteq G \times E \tag{4.31}$$

Since a directed acyclic graph is needed, it is also necessary to prohibit cycles. In order to do so a relation needs to be defined as transitive closure (Equation 4.24).

$$hasSubset^* \subseteq G \times E \tag{4.32}$$
$$\begin{aligned}\forall g \in G \, \forall e \in E \, : (g, e) \in hasSubset^* \\ \rightarrow (g, e) \in hasSubset \\ \vee (\exists e' : (g, e') \in hasSubset^* \\ \wedge (e', e) \in hasSubset)\end{aligned} \tag{4.33}$$
$$\forall g \in G \, \forall e \in E \, : (g, e) \in hasSubset^* \rightarrow (e, g) \notin hasSubset^* \tag{4.34}$$

Based on this, it is possible to prohibit cycles such as it is done by Equation 4.34. At this point the desired structure has been defined. Furthermore, due to the fact that Tenants were defined to be entities as well, it is possible to assign Tenants to Groups at any Level.

Definition of Dimensions

The final step to conclude the modeling of the Dimension, Group, and Tenant structure is the modeling of Dimensions. The set has already been defined in Section 4.4.2. The relation that allows to associate Groups to Dimensions is called *isRealizedBy*. Equation 4.35 defines the relationship as a function that assigns a set of Groups to any given Dimension.

$$isRealizedBy \subseteq D \times G \tag{4.35}$$

It was stated in Section 4.2.4 that every Group may only belong to a single Dimension. This may be the case either directly associated by a $isRealzedBy$-relationship or transitively by being a Subgroup of Groups that has a direct $isRealzedBy$-relation. This is expressed by the Equations 4.37 and 4.38.

$$isRealizedBy^* \subseteq D \times G \tag{4.36}$$
$$\begin{aligned}\forall g \in G \, \forall d \in D : (d, g) \in isRealizedBy^* \\ \rightarrow (d, g) \in isRealizedBy \\ \vee (\exists g' : (d, g') \in isRealizedBy \\ \wedge (g', g) \in hasSubset^*)\end{aligned} \tag{4.37}$$
$$\forall g \in G \rightarrow \exists! d \in D : (d, g) \in isRealizedBy^* \tag{4.38}$$

The final constraint that needs to be defined is the aforementioned requirement that every Tenant needs to be associated to at least one Group per Dimension. This is defined by Equation 4.39.

$$\forall t \in T\, \forall d \in D \rightarrow \exists\, g \in G : (g, t) \in hasSubset \wedge (d, g) \in isRealizedBy^{*} \tag{4.39}$$

4.4.6 Description of Deployment Models

In Section 4.2.2 and Section 4.3.3 the way Customers shall have to define their Deployment Constraints was discussed. Since this also needs to be captured by the Description Model, it is defined by the following paragraphs.

Deployment Constraint Structure

In Section 4.4.2 it was defined that a Deployment Constraint needs to be associated to exactly one Tenant, one or many Application Components, and one or many Deployment Levels. Furthermore, a Deployment Constraint also needs to be of a specific Deployment Model. The Deployment Models that shall be supported by the Description Model were discussed in Section 4.2.2. In order to be able to realize this, the sets PR, PU, WH, BL, and GR were defined. These sets shall be treated as Deployment Constraints. This is expressed by equation 4.40.

$$\forall dc : dc \in PR \oplus dc \in PU \oplus dc \in WH \oplus dc \in BL \oplus dc \in GR \leftrightarrow dc \in DC \tag{4.40}$$

Furthermore, it was stated in Section 4.2.2 that there shall not be any other Deployment Models supported but the mentioned five ones. Thus, the Constraint 4.40 also states that every individual, that is of type Deployment Constraint, must also be of the type of one of the Deployment Models. The usage of the proposed inheritance structure avoids redundancy.

includes-Relationship

In Section 4.2.2 it was stated that the Deployment Models *White Hybrid* and *Gray Hybrid* shall have the ability to include Groups or Tenants. This means that the Customer specifies that they would feel comfortable to share with these Groups or Tenants. Thus, it is necessary to define the relationship *includes* as a relation that may be used between a *White Hybrid* or a *Gray Hybrid* Deployment Constraint and an entity. This is expressed by Equation 4.41. Furthermore, Equation 4.42 defines that a Deployment Constraint (either *White Hybrid* or *Gray Hybrid*) may include one to many entities.

$$includes \subseteq (WH \times E) \cup (GR \times E) \tag{4.41}$$

$$\forall dc \in DC : dc \in WH \vee dc \in GR \rightarrow \exists\, e \in E : (dc, e) \in includes \tag{4.42}$$

However, this does not define the entire functionality the Description Model was supposed to capture. Section 4.2.4 stated that if a particular Group is being included, all Groups and Tenants will be included that are a subset of this Group.

This may be expressed by defining the transitive closure for the *includes*-relation. This is done by Equation 4.44.

$$includes^* \subseteq (WH \times E) \cup (GR \times E) \tag{4.43}$$

$$\begin{aligned} \forall e \in E \, \forall dc \in DC : (dc \in WH \vee dc \in GR) \wedge (dc, e) \in includes^* \\ \rightarrow (dc, e) \in includes \\ \vee (\exists e' \in E : (dc, e') \in includes \\ \wedge (e', e) \in hasSubset^*) \end{aligned} \tag{4.44}$$

excludes-Relationship

Similar to the just discussed *include*-relationship, the *exclude*-relationship may be defined. It was discussed in Section 4.2.2 that the *exclude*-relationship is necessary for the Deployment Models *Black Hybrid* and *Gray Hybrid*. It may be used to exclude Tenants or Groups. Thus, it is defined as follows.

$$excludes \subseteq (BH \times E) \cup (GR \times E) \tag{4.45}$$

$$\forall dc \in DC : dc \in BH \vee dc \in GR \rightarrow \exists \, e \in E : (dc, e) \in excludes \tag{4.46}$$

Similar to the *includes*-relationship, the *excludes*-relationship also works transitively through the Group structure. This may be expressed using the following Constraint.

$$excludes^* \subseteq (BH \times E) \cup (GR \times E) \tag{4.47}$$

$$\begin{aligned} \forall e \in E \, \forall dc \in DC : (dc \in BH \vee dc \in GR) \wedge (dc, e) \in excludes^* \\ \rightarrow (dc, e) \in excludes \\ \vee (\exists e' \in E : (dc, e') \in excludes \\ \wedge (e', e) \in hasSubset^*) \end{aligned} \tag{4.48}$$

4.4.7 Completeness of Deployment Constraints

So far many constraints have been defined that define the Description Model's structure. But there is one constraint left that needs to be defined in order to ensure that complete information is available to extract Deployment Information. Equation 4.49 defines that there needs to be a Deployment Constraint created for every Tenant, every Application Component, and on all relevant Deployment Levels.

$$\begin{aligned} \forall t \in T \, \forall ac \in AC \, \forall dl \in DL : (ac, dl) \in isDeployedOn \\ \rightarrow \exists! \, dc \in DC : (dc, ac) \in of \wedge (dc, t) \in for \wedge (dc, dl) \in onLevel \end{aligned} \tag{4.49}$$

If this constraint would not be true, the Deployment Model would not capture all necessary information. Due to incomplete data, it would not be possible to determine how a specific Application Component needs to be deployed.

4.4.8 Definition of the Deployment Information

Based on the definitions so far, it is now possible to define the Deployment Information. Before doing so, in a first step a new set of sets shall be created that allows to define all following constraints more easily. This set of sets is, in fact, an alternative representation of Deployment Levels. In this new definition each Deployment Level is not defined as a simple entity but as a set of Application Components that are deployed on this Deployment Level. This may be defined as follows.

$$\mathcal{DLS} = \{DLS_{dl_1} \mid DLS_{dl_1} = AC\} \cup \{DLS_{dl_i} \subseteq AC \mid 2 \leq i \leq idl\} \tag{4.50}$$

$$\forall ac \in AC \quad \forall i \in \{2, \dots, idl\} : ac \in DLS_{dl_i} \rightarrow (ac, dl_i) \in isDeployedOn \tag{4.51}$$

Based on this, it is possible to define the Deployment Information more easily.

It has previously been discussed (Section 4.3.3) that the Deployment Information may very well be expressed as graphs. Thus, it is defined as a set of undirected graphs as follows.

$$\mathcal{DI} = \{DI^{ac}_{dl} = (V^{ac}_{dl}, E^{ac}_{dl}) \quad | \quad dl \in DL \wedge ac \in DLS_{dl}\} \tag{4.52}$$

$$\forall dl \in DL \quad \forall ac \in DLS_{dl} : V^{ac}_{dl} = T \wedge E^{ac}_{dl} \subseteq T \times T \tag{4.53}$$

As stated earlier, if two Tenants share an edge in a given graph, these two Tenants are allowed to share a Unit of the Application Component and the Deployment Level this graph applies to.

In order to have complete Deployment Information, it is necessary to define additional trivial Deployment Constraints. The first one is that each Tenant must always be allowed to share with themselves. The second is that if there is an edge between two Tenants, they must both be allowed to share with each other. These are very obvious constraints that are defined by making the set of edges reflexive and symmetric.

$$\forall dl \in DL \quad \forall ac \in DLS_{dl} \quad \forall t \in T : (t, t) \in E^{ac}_{dl} \tag{4.54}$$

$$\forall dl \in DL \quad \forall ac \in DLS_{dl} \quad \forall (t, t') \in E^{ac}_{dl} : (t', t) \in E^{ac}_{dl} \tag{4.55}$$

The last required constraint was discussed in Section 4.3.3. This section defined that lower Level Deployment Constraints must be equally or less restrictive than their predecessors. This is due to the structure of a Valid Deployment. If two Tenants share a Unit of the instance Level, for example, they must also share all underlying Units like the virtual machine. This constraint is formalized by the following.

$$\forall dl, dl' \in DL \quad \forall ac \in DLS_{dl} \cap DLS_{dl'} : dl < dl' \rightarrow E^{ac}_{dl} \subseteq E^{ac}_{dl'} \tag{4.56}$$

Based on these basic definitions of the Deployment Information, it is now possible to describe how the constraints that are defined in the Description Model relate to the Deployment Information. It has previously been stated that two Tenants may only share Units if neither of them has a Deployment Constraint prohibiting it (Section 4.2.2 and 4.3.3). This means, more specifically, that two Tenants may only share a given Unit on a given Deployment Level if for both Tenants the following rules apply.

Private Neither Tenant shall have chosen the private Deployment Model.

White Hybrid Neither Tenant shall have chosen a white hybrid Deployment Model without included the other Tenant. If a Tenant would have used the white hybrid Deployment Model and included the other, sharing would be allowed (assuming that the other Tenant also expressed a constraint that permits sharing).

Black Hybrid None of the Tenants shall have chosen a black hybrid Deployment Model and excluded the other Tenant. If a Tenant would have used the black hybrid Deployment Model and not excluded the other Tenant, sharing would be allowed (assuming that the other Tenant also expressed a constraint that permits sharing).

Gray Hybrid None of the Tenants shall have chosen a gray hybrid Deployment Model where the other Tenant is not included or is excluded. If a Tenant would have used the gray hybrid Deployment Model but would have included and not excluded the other Tenant, sharing would be allowed (assuming that the other Tenant also expressed a constraint that permits sharing).

All this is expressed by the following constraint.

$$
\begin{aligned}
&\forall dl \in DL \quad \forall ac \in DLS_{dl} \quad \forall t, t' \in T : (t, t') \in E_{dl}^{ac} \\
&\quad \rightarrow \nexists dc \in DC : (dc, dl) \in onLevel \wedge (dc, ac) \in of \\
&\qquad \wedge \Big[(dc, t) \in for \wedge (dc \in PR) \\
&\qquad\quad \vee (dc \in WH \wedge (dc, t') \notin includes^*) \\
&\qquad\quad \vee (dc \in BH \wedge (dc, t') \in excludes^*) \\
&\qquad\quad \vee (dc \in GH \wedge (dc, t') \in excludes^* \wedge (dc, t') \notin includes^*) \\
&\qquad \Big] \vee \Big[(dc, t') \in for \wedge (dc \in PR) \\
&\qquad\quad \vee (dc \in WH \wedge (dc, t) \notin includes^*) \\
&\qquad\quad \vee (dc \in BH \wedge (dc, t) \in excludes^*) \\
&\qquad\quad \vee (dc \in GH \wedge (dc, t) \in excludes^* \wedge (dc, t) \notin includes^*) \Big]
\end{aligned} \tag{4.57}
$$

4.5 Prototypical Realization

In the previous sections the requirements towards the Description Model and the Utilization Process have been analyzed. Based on this, the last section presented a technology-independent, formal Deployment Model that may be used as a blueprint for implementation.

This section's purpose is to describe a realization that was created as part of this work. The implementation is based on using Semantic Web Technologies. The advantage of using these particular technologies is that some parts of the Description Model may very easily be realized by expressing them as an ontology and applying a piece of software, a so called semantic reasoner, that is able to infer logical consequences from the description. Furthermore, these technologies provide a very easy and straight forward way to extract Deployment Information.

Thus, the section is structured as follows. Before introducing the realization, the Section starts with a selection of a suitable realization technique (Section 4.5.1). Based on this, Sections 4.5.2 and 4.5.3 introduce the prototypical realization of the Deployment Model. This Section concludes by describing how the Deployment Information may be extracted from the implemented Deployment Model (Section 4.5.4).

The results of this section were created in cooperation with Holger Wache. We also published the results jointly in a paper [RWV13].

4.5.1 Description of Realization Approach

For the realization of the proposed Description Model there are many approaches possible. The most obvious one is the implementation using standard programming languages and a relational database management system to store the model. Using this approach, it would be possible to capture the different entities and their basic relationships using standard techniques of relational databases [Saa10]. However, more complex relationships such as the defined transitive including and excluding of Tenants and Groups of Tenants would need to be implemented manually using a standard programming language.

Another approach would be to use a knowledge representation language that allows not just to describe the entities and their basic relationships, but also to model complex relationships. Thus, implementation effort can be minimized. Due to this fact, the Web Ontology Language (OWL) has been used for the implementation. OWL is a family of knowledge representation languages for authoring ontologies. It was specified by the W3C [W3C12] and is used in academia and the medical field as well as commercially [AH11].

One additional advantage of using OWL is that there are many libraries, tools, and documentations available, which allow to use it efficiently. However, there is a problem that comes with the usage of OWL. OWL utilizes the Open World Assumption. An open world is one where it is assumed that at any time new information could come to light and new conclusions may be drawn from it [RN10, pg. 417][AH11]. The problem with this is that at some points, it is critical for the Description Model that the total knowledge is captured since complete Deployment Information shall be extracted. However, it is quite common to encounter this problem when OWL models shall be utilized in an application. The usual approach to tackle this problem is to manually close the world at the points where this is necessary. For this prototype this is done by implementing a model checker which will be introduced in Subsection 4.5.3.

Besides OWL, the following other technologies and utilities were used for the implementation of the prototype.

SPARQL Protocol And RDF Query Language (SPARQL) SPARQL is a query language for databases, able to retrieve and manipulate data stored in a Resource Description Framework format [W3C13]. For the prototype it is used to extract the Deployment Information from the Description Model. Details will be discussed in Section 4.5.4.

Semantic Web Rule Language (SWRL) SWRL is a proposed language for the Semantic Web that can be used to express rules as well as logic [W3C04]. It is used to extract the Deployment Information from the OWL model. This will be further elaborated in Section 4.5.4.

Protégé Protégé is an ontology editor and knowledge-based framework [pro13]. Protégé has been used to create the OWL model of the Description Model.

Jena-Framework It is a Java framework that allows to build Semantic Web applications [Apa13]. In order to do so, it provides a collection of tools and Java libraries. For the prototype it has been used to access, compute, and reason based on the OWL model.

GraphStream Library GraphStream is a Java library that allows to model and analyze graphs [The13]. For the Description Model the library has been used to store and visualize the Deployment Information.

Java Finally, the general-purpose programming language Java has been used for the implementation of the prototype. The primary reason for this is the availability of suitable third-party libraries and its platform-independent nature.

4.5.2 Implementation of the Model using OWL

In Section 4.4 the Deployment Model was proposed which is supposed to be able to allow to capture Customers' Deployment Constraints. Figure 4.7 gives an overview of the Description Model's realization using OWL. The following discusses the OWL model in more detail. Most of the elements that the Description Model consists of, directly correspond to their counterparts in the Description Model of Section 4.4.2. The following are the classes that were created in OWL.

AC represents the Application Components the application consists of (corresponds to set AC).

Level represents the Deployment Levels or the System (corresponds to set DL).

Deployment Constraint - Private, Public, White, Black, Gray Deployment Constraint (set DC) represents a generic Deployment Constraint. Private

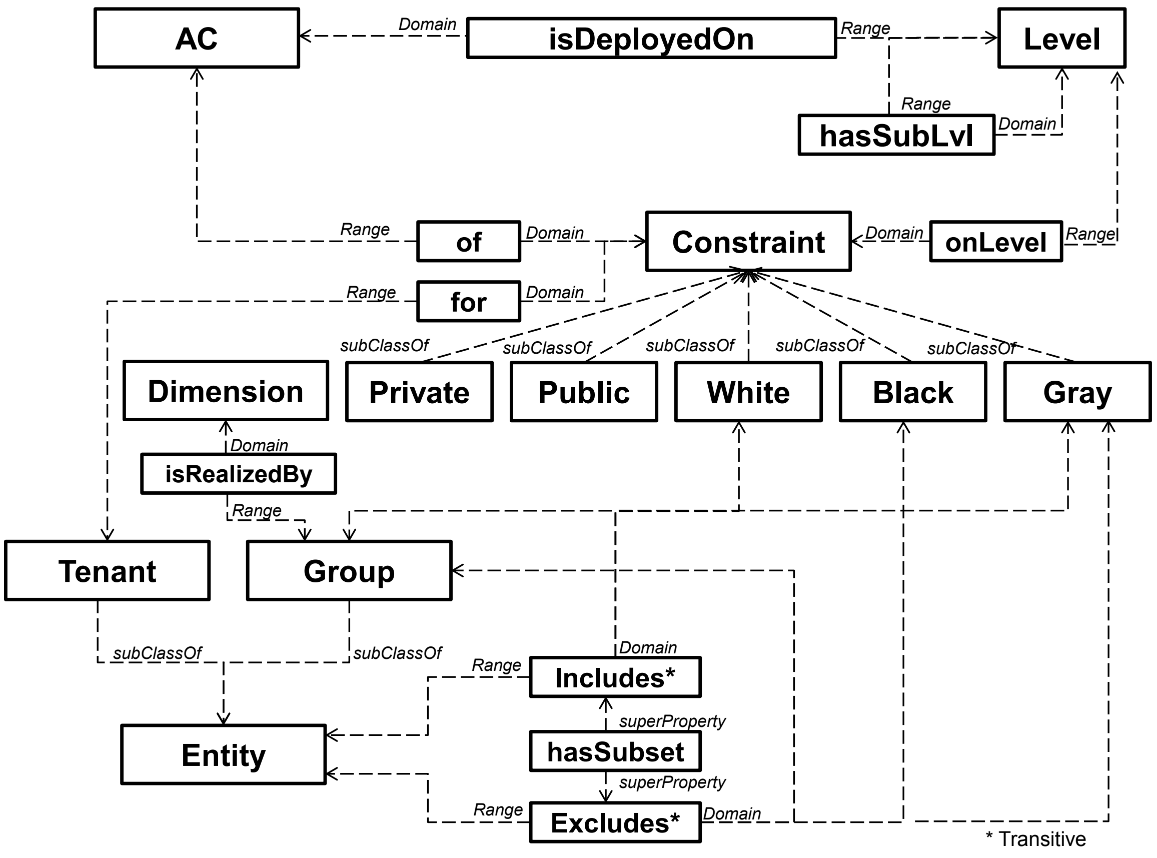

Figure 4.7.: Realization of the Description Model using OWL (incomplete)

(set Pr), Public (set Pu), White (set Wh), Black (set Bl), and Gray (set Gr) represent the different Deployment Models that shall be supported by the system. Furthermore, the Description Model defined (Constraint 4.40) that each of the five Deployment Models shall also be of type Deployment Constraint. This was realized by a *subClassOf*-property in the OWL model which indicated inheritance. Furthermore, it was defined that every Deployment Constraint shall only have exactly one Deployment Model as type. This was realized using OWL by defining each of the five Deployment Levels as disjoint classes to each other. In the Description Model it was stated that a Deployment Constraint shall be defined as abstract - meaning that there shall not exist instances of this type. It is not possible to express such a Constraint in OWL. Thus, this will be realized outside of the OWL model [Rus07]. A description of this realization is given in Section 4.5.3.

Entity, Tenant, Group Entity (set E defined by Constraint 4.29), Tenant (set T), and Group (set G) directly correspond to their counterparts in the Deployment Model. Those sets were defined to capture the structure between Tenants and Groups. For this the composite pattern ([GHJ94]) was utilized. Similar to the deployments, Constraint 4.30 defined to have an inheritance structure between the *Entity*-class and the classes *Tenant* and *Group*. Again this was realized using the *disjoint* and the *subClassOf*-property. The definition of abstract will be discussed in Section 4.5.3.

Furthermore, the OWL model consists of the following object properties.

of is a relationship (Constraint 4.12) that associates Application Components to a Deployment Constraint. Thus, it has *Deployment* as domain and AC as range. It was defined that each Deployment Constraint has one to many Application Components associated (Constraint 4.13). This has been implemented in OWL using the cardinality restriction. However, due to the open world assumption, the OWL reasoner will not check this Constraint. How the desired behavior can still be gained will be discussed in Section 4.5.3.

for implements the Constraints 4.14. The *for*-property associates Tenants to a Deployment Constraint. Thus, its domain is *Deployment* and range *Tenant*. In addition, Constraint 4.15 defined that a Deployment Constraint may only be associated to exactly one Tenant. This is realized using cardinality restriction. Due to the open world assumption, special means are necessary to ensure that at least one Tenant is associated (introduced in Section 4.5.3). However, to ensure that not more than one is associated, it is necessary to define all Tenants as distinct from each other. If this definition would not be made, the reasoner would determine that all Tenants associated are in fact equal. By introducing the distinct feature the reasoner will be forced to indicate an inconsistency in the OWL model if more than one Tenant is associated.

onLevel associates Deployment Constraint to the Deployment Levels they shall apply to. That is why its domain is *Deployment* and its range *Level* as defined by Constraint 4.16. Constraint 4.17 defined that a *Deployment* may apply to one to many Levels. Again this is implemented using the cardinality restriction and the check that will be described in Section 4.5.3.

isDeployedOn is defined to associate Deployment Levels to Application Components (Constraints 4.18). Thus, it has the domain *Deplyoment* and the range *Level*. For the *isDeployedOn*-relation it was defined by 4.19 that every AC has at least one DL associated. This is covered by OWL cardinality restrictions and further by the technique that will be introduced in Section 4.5.3.

isRealizedBy is supposed to allow to associate *Groups* to *Dimensions* (Constraint 4.35). Thus, it is defined with *Dimension* as domain and *Group* as range. Furthermore, it needs to be defined as transitive (Constraint 4.37).

hasSubLvl is used to create directed graphs for the structure of DLs (Constraint 4.22). Thus, domain and range are defined to be the *Level*-class. Furthermore, a transitive closure was defined (Constraint 4.24). In the OWL model this was captured by a new second relationship (called hasSubLvlTrans) that is defined as transitive. It is super property to the regular relation. This pattern is described by [AH11, pg. 164] and applies very well to this OWL model. It allows to query for both, direct associations and transitive ones, based on only one description.

hasSubset has the purpose of modeling the structure between *Group*s and *Tenant*s as graphs in which Tenants will always be leafs (Constraint 4.31). According to this definition the relation has the domain *Group* and the range *Entity*. Constraint 4.32 defined a transitive closure for this relationship. Due to this, this relationship is defined as transitive in the OWL model.

includes and excludes were defined as relations that allow Deployment Constraints to exclude or include Groups or Tenants (Constraints 4.41 and 4.45). Thus, both properties have the *Entity*-class as range. The domain differs depending on the relationship. It was said that only the WhiteHybrid and the GrayHybird Deployment Model shall be able to include Groups or Tenants. Thus, the domain for includes is *White* or *Gray*. Furthermore, since only the Black Hybrid and the Gray Hybrid Deployment Model shall be able to exclude Groups and Tenants, *Black* or *Gray* are domain to the *excludes*-relationship.

With respect to cardinalities, it was stated that every Deployment Constraint, that is of type White Hybrid, Black Hybrid, or Gray Hybrid shall have at least one include and/or exclude (Constraints 4.42 and 4.46). This was realized by cardinality restrictions and the checking mechanism that will be introduced in Section 4.5.3.

Both, *includes* and *excludes*, are defined as super property to the *hasSubSet*-property. Thus, every usage of *hasSubset* will also result in the creation of an exclude and an include. This may then be used to include or exclude Groups (or Tenants) based on reasoning. Due to this, it is necessary that *includes* and *excludes* have *Entity* as an additional domain. Otherwise, the proposed approach would not work since every associated *Entity* would be reasoned to be one of the Deployment Models.

The final step to define these two properties completely is to define them as transitive (Constraint 4.44 and 4.48). Thus, it is possible to include or exclude Groups and Tenants down the *hasSubSet*-structure.

4.5.3 Model checking

Due to the fact that OWL is a language that uses the open-world assumption, it was not possible to implement all constraints that were defined in Section 4.4. However, since it is the goal of this work to provide a complete implementation of the Deployment Model, it is necessary to close the world at certain points. This goal is achieved through the implementation of a model check.

The model checker allows to check if the model applies to even those constraints that were not expressible in OWL and those that require a closed-world assumtion. It was implemented in Java using the Jena library and SPARQL. In case the check is not all valid, the OWL model would not be considered valid and a Deployment Information extraction would not be performed.

The following checks were implemented in order to cover the entire Description Model completely.

OWL-Model is in a valid state The purpose of the model checker is to determine if the OWL model is in a valid state. This is why the first check determines if all constraints apply that were defined in the previous subsection. This is done by having the OWL-reasoner check the OWL model and check if it contains conflicts.

Constraints and Entity are defined abstract In Section 4.4 it was stated that the classes Constraint and Entity shall be abstract (Constraints 4.40 and 4.30) - meaning that no instances of these classes shall exist. OWL does not provide a mechanism to create an abstract class. Thus, it is necessary to check if instances of the two classes exist. If so, the OWL model is not valid.

Check of cardinalities At the beginning of the Realization section it was stated that OWL is based on an open-world assumption. For the object properties *of*, *for*, *onLevel*, *isDeployedOn*, *isRealizedBy*, *include*, and *exclude*, it was stated that there shall be at least (or exactly) one instance of another class associated. However, due to the open-world assumption, the reasoner will not highlight an error if such an instance does not exist in the described OWL model. Thus, this is the task of this checker.

Deployment Levels-Hierarchy is free of cycles The structure of DLs was discussed to be a directed acyclic graph. Cycle freeness was defined by Constraint 4.25. In the previous Section the *hasSubLevel*-Relationship was created which allows to define directed graphs. However, similar to the previous check, this check evaluates if the structure is free of cycles.

Level dl_1 is the highest Level in Deployment Level-Hierarchy It has previously been defined that Level dl_1 must be the highest Level in the Deployment Level hierarchy (Equation 4.28). This is ensured by this check.

All Application Components are associated to Deployment Level dl_1 According to Equation 4.19 all Application Components must be deployed on Level dl_1. This is ensured by this check.

All Application Components are associated only to Deployment Levels of the same Stack It was defined that an Application Component shall only be deployed on Deployment Levels that belong to the same stack (Constraints 4.26 and 4.27). Since it was not possible to express this with OWL, this Constraint is examined by this check.

The *Entity-Group*-Hierarchy is free of cycles Constraint 4.34 defined that the *Entity-Group*-Hierarchy shall be a directed acyclic graph. In the previous section the *subSetOf*-Relationship was introduced to create directed graph structure. This check proves that the structure is free of cycles.

Every Group belongs to exactly only oneDimension It was discussed that every Group shall be directly or transitively associated to exactly one Dimension. This was defined by Constraint 4.38. Thus, this check determines if the constraint is met.

Every Tenant is associated to one Group per Dimension It was discussed that Tenants shall be associated to at least one Group per Dimension (Constraint 4.39). Whether this constraint is met is determined by this check.

Every Constraint is associated to valid combinations of Levels and Application Components Equation 4.21 defined that a Deployment Constraint may only be associated to those combinations of Application Components and Deployment Levels, where at least one Application Component is actually deployed on the Deployment Level. This is checked here.

Every Tenant has exactly one Constraint defined for every Application Component, DL In order to have an unambiguous description of Tenants' Deployment Constraints, it is necessary to ensure that there is only one deployment defined per Application Component, Deployment Level, and Tenant. This was defined by Constraint 4.49. This check ensures just that.

4.5.4 Extraction of Deployment Information

The information necessary for the next step (discussed in Chapter 5) is the Deployment Information. It captures whether two Tenants are allowed to share Deployment Units of the same Application Component at the same Deployment Level. A feasible representation of this information is an undirected graph per combination of Application Component and Deployment Level. In these graphs, nodes represent Tenants and edges represent if they may share infrastructure, as this has been discussed in Section 4.3.4. The information may be extracted from the OWL model with the help of a two-step process.

The first step is to introduce an abstract representation of all requirements by relating Tenants (and not only Groups) to Deployment Constraints directly where they are not explicitly excluded. Then, in a second step, a list of Tenant-pairs are retrieved saying these Tenants may share the same resources.

Step 1: Introduction of "mayDeployTenant"-Property

The first step toward extraction of the Deployment Information is to introduce one additional property called *mayDeployTenant* to the OWL model. Its purpose is to associate the Tenants directly to every Deployment Constraint which may be deployed together.

This additional property is not simply the closure of the transitive property *includes* but also requires to remove those Tenants and Groups which are excluded – depending on the Deployment Model. Therefore, the following rules are described in a SWRL-like notation of how *mayDeployTenant* is introduced into the OWL model.

Assigning the Owner Any Deployment Constraint may always deploy the owner. This may be expressed by the following rule: Deployment(?d) $\wedge$ Tenant(?t) $\wedge$ for(?d,?t) $\rightarrow$ mayDeployTenant(?d, ?t)

Private A private deployment may only deploy the owner. Since this has been taken care of with the previous rule, there is nothing more to add at this point.

Public A public deployment may deploy any Tenant that is using the application. Thus[4]: Public(?d) $\wedge$ Tenant(?t) $\rightarrow$ mayDeployTenant(?d, ?t)

White Hybrid A white hybrid deployment may only deploy Tenants that are explicitly included. This rule uses the closure of the transitive property *includes*: WhiteHybrid(?d) $\wedge$ Tenant(?t) $\wedge$ includes(?d, ?t) $\rightarrow$ mayDeploy-Tenant(?d, ?t)

Black Hybrid A black hybrid deployment may deploy all Tenants that are not explicitly excluded: BlackHybrid(?d) $\wedge$ Tenant(?t) $\wedge$(not excludes(?d, ?t)) $\rightarrow$ mayDeployTenant(?d, ?t)

Gray Hybrid A gray hybrid deployment may only deploy Tenants that are explicitly included and not excluded. GrayHybrid(?d) $\wedge$ Tenant(?t) $\wedge$ includes(?d, ?t) $\wedge$ (not excludes(?d, ?t)) $\rightarrow$ mayDeployTenant(?d, ?t)

In the last two rules the negation "not" is used to indicate that the property *excludes* (and its transitive closure) cannot be derived between ?d and ?t. However, this form of negation corresponds to closed-world assumption, i.e. the "not" is used as negation-as-failure. Because OWL follows the open-world assumption, the negation cannot be implemented as OWL inferences. Therefore, these rules are implements with the help of Jena and SPARQL. The (closed) pairs of "excludes(?d, ?t)" are retrieved with the help of a SPARQL query and subtracted from the positive pairs between ?d and ?t (retrieved with a second SPARQL query).

Step 2: SPARQL-Queries to extract graph

Once the *mayDeployTenant*-Property has been inserted in the OWL model, the desired Deployment Information of a given Application Component `*AC*` and level `*L*` may be extracted by using the following SPARQL-Query.
SPARQL Query to extract Deployment Information

```
PREFIX ns: < nameSpace >
SELECT ?t1 ?t2 WHERE {
   ?t1 a ns:Tenant .
   ?t1 ns:isCustomer true .
   ?t2 a ns:Tenant .
```

[4]This rule is not a mistake. Because of the definition of a public Deployment Model, *any* Tenant can be deployed to *any* public deployment.

```
   ?t2 ns:isCustomer true .
   ?deplyomentsOfComponent1 ns:deploymentOf *AC* .
   ?deplyomentsOfComponent2 ns:deploymentOf *AC* .
   ?deplyomentsOfComponent1 ns:deploymentFor ?t1 .
   ?deplyomentsOfComponent2 ns:deploymentFor ?t2 .
   ?deplyomentsOfComponent1 ns:mayDeployTenant ?t2 .
   ?deplyomentsOfComponent2 ns:mayDeployTenant ?t1 .
   ?deplyomentsOfComponent1 ns:onLevel *L* .
   ?deplyomentsOfComponent2 ns:onLevel *L*
}
```

The query returns a list of two Tenants per line, similar to this one: *SPARQL Query to extract Deployment Information*

```
-------------------
| t1     | t2     |
===================
| ns:T-2 | ns:T-2 |
| ns:T-2 | ns:T-1 |
| ns:T-1 | ns:T-2 |
| ns:T-1 | ns:T-1 |
...
-------------------
```

If two Tenants are represented in one line, this means that they may be deployed together. It does that with no redundancy. This means if Tenant A and Tenant B may be deployed together, the query will only return A → B and not B → A in addition. This line is only gained if both Tenants have agreed to share infrastructure. In order to be sure that even Tenants are gained that have no connections to other Tenants, every Tenant will appear at least once associated to themselves (A → A).

4.6 Evaluation

This section's purpose is to discuss and evaluate the presented approach of capturing Customers' Deployment Constraints towards Mixed-Tenancy Deployments. This is done by demonstrating that everything that was supposed to be describable (according to requirements analysis of Section 4.2) is describable. It will do so by creating an example that is based on the requirements analysis, that represents the full range of what shall be describable. Once created, the scenarios will be modeled and Deployment Information will be extracted using the prototype tool that was discussed in the previous Section (Section 4.5).

In order to accomplish this, the section is structured as follows. It starts (Section 4.6.1) with a definition of cases for the requirements formulated in Section 4.2. Based on those representative, scenarios are created and the Deployment Information is extracted. This is discussed in Section 4.6.2. The final Section 4.6.3 will report the results.

4.6.1 Example Environment

The purpose of this subsection is to create an example environment that contains many cases that shall be describable according to the discussion of Section 4.2. Based on the environment introduced here, the next section will define three Scenarios how Customers may express their Deployment Constraints.

Deployment Level Hierarchy

In Section 4.2.1 it was defined that it is possible that not all Application Components that compose an application have the same Deployment Level stack. In fact, it was stated that all Deployment Levels of an application shall be describable as a directed acyclic graph. A single Application Component, however, shall only be associated to a stack of Deployment Levels that are in this graph. For this evaluation, the Deployment Level Graph illustrated by Figure 4.8 has been created. In the example there is one highest Deployment Level, called instance.

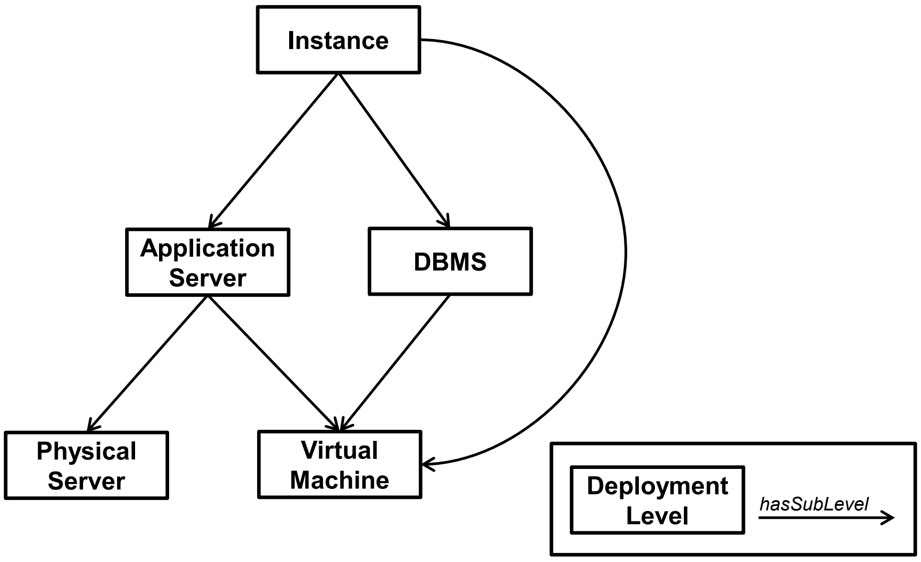

Figure 4.8.: Evaluation Scenario that Includes all Possibilities

This is due to the definition of Section 4.2.1. In the graph, there are multiple lowest-level Deployment Levels and multiple alternative stacks. Furthermore, there are cases in which a Deployment Level has multiple successors and those where there is only one.

Based on the graph it is possible to associate Application Components to the following Deployment Level stacks:

1. Instance - Application Server - Physical Server
2. Instance - Application Server - Virtual Machine
3. Instance - DBMS - Virtual Machine
4. Instance - Virtual Machine

Application Components

Based on the definition of these Stacks, it is now possible to associate a set of Application Components to the stacks they shall be deployed on. It is important that every Application Component may only be deployed on one stack. For the example there are stacks that are only used by a single Application Component and others that are used by multiple ones. How this is done for the example is illustrated by Table 4.2. For this example a total of five Application Components

Stack	Application Component
1	AC-2
2	AC-3, AC-4
3	AC-1
4	AC-5

Table 4.2.: Mapping between Stacks and Application Components

is sufficient to have multiple cases which associate one or multiple Application Components to the same stack.

Dimension, Group Hierarchy

In Sections 4.2.3, 4.2.4, and 4.4.5 it was discussed that it shall be possible to create multiple Dimensions. Each Dimension shall be realized by a set of Groups. Furthermore, it shall be possible that Groups have Subgroups. In fact, it was defined that each Dimension shall be the root of a directed acyclic graph. For the example the Dimensions and Groups illustrated by Figure 4.9 were created. In

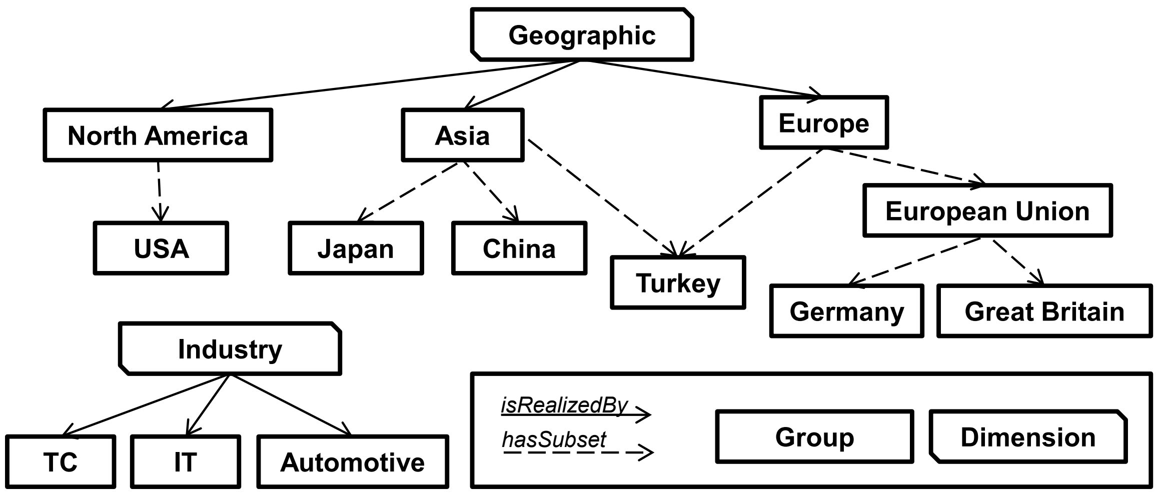

Figure 4.9.: Definition of Example Dimensions and Groups

this example two Dimension are created.

Geographic Defines in which country a Tenant has its head quarters. It has a complex structure of underlying Groups that form a directed acyclic graph.

Industry Determines in which industries a Tenant does business. This Dimension has only one level of Groups that have no structure.

Tenants

In Section 4.2.5 it was analyzed that there shall be two types of Tenants captured by the model, Tenants and Virtual Tenants. The difference between them is that Tenants are Customers of the application and Virtual Tenants are placeholders for Tenants that are needed for Deployment Constraint definition. For the example both need to be considered in order to demonstrate the expressiveness of the model.

According to the definition in Section 4.4.5, it is necessary to associate each Tenant to at least one Group per Dimension. Table 4.3 depicts how this is done for the example. Please note that there are cases in which a Tenant is associated to only one Group per Dimension and a case where they are associated to multiple ones.

Tenant	Geographic	Industry
T-A	USA	IT
T-B	Japan	Automotive
T-C	China	Automotive
T-D	Turkey	IT
T-E	Germany	IT, TC
T-F	Great Britain	TC

Table 4.3.: Mapping between Tenants and Groups

4.6.2 Scenario Definition

Based on the environment defined in the previous section, this section defines three scenarios how Customers may define their Constraints. The first two are quite simple. The idea of them is to analyze both extremes of how resources may be shared. Based on this, the third is defined to be a mixture of both.

Scenario 1: All Private In the first scenario, all Customers select the private Deployment Model for all Application Components on all Deployment Levels. This means that each Tenant states that they require their own Unit of all Application Components and all Deployment Levels. In the resulting Deployment Information, there should not be any edges between nodes in any graph.

Scenario 2: All Public In the second scenario, all Customers select a Public Deployment Model for all Application Components and all Deployment Levels. This means that each Tenant states that they share all Application Components and all Deployment Levels with all other Tenants. All graphs of the Deployment Information should be complete.

Scenario 3: Mix In the third scenario all possible Deployment Models are used. In this scenario each Tenant has different requirements towards the sharing of resources.

For the first and second scenario, it is not necessary to give any additional description as it has already been described completely by the simple introduction. For scenario 3, however, this is different. Here, every Tenant expresses different Deployment Constraints.

Tenant T-A has no Constraints towards sharing at all, thus, everything is public. Tenant T-F, on the other hand, is very conscious about sharing, thus, does not share at all. The other Tenants express their Deployment Constraints somewhere between those extremes. In general, it is assumed that with ascending numbers Tenants get more restrictive. Application Components, on the other hand, are assumed to get less crucial with ascending numbers. Table 4.4 illustrates the Deployment Constraints expressed by all Tenants in detail.

4.6.3 Analysis of Results

Based on the three scenarios, it is possible to extract the Deployment Information that will be used as an input for the next step of computing a Valid and Optimal Deployment. It has been stated before that the Deployment Information contains an undirected graph for every Application Component and Deployment Level for which the Application Component is deployed on it. Each of these graphs expresses which Tenants are allowed to share the Application Component on a particular Level. For the three scenarios that means the following.

Scenario 1: All Private The Deployment Information of this Scenario will only contain graphs entirely without edges. This is due to the fact that none of the Tenants agreed to share Units at all.

Scenario 2: All Public The Deployment Information of this Scenario will only contain complete graphs. This is due to the fact that all Tenants agreed to share with all other Tenants by using only the Public Deployment Model.

Scenario 3: Mix In this Scenario the graphs contained by the Deployment Information are not that easily described. Thus, Figure 4.10 visualizes the Deployment Information of this scenario.

It has previously been defined that only then two Tenants will be adjacent in a graph if both have not expressed Constraints that prohibit sharing. This is also visible in the example, for example, Tenant T-F. They defined to use everything privately. Thus, this Tenant is not adjacent to any other Tenant in any other graph, even though other Tenants (e.g. Tenant T-A) have expressed that they would be willing to share with Tenant T-F.

Concluding the examples, it may be said that everything that was supposed to be capturable by the Description Model, according to the requirements analysis of Section 4.2, was capturable in this example.

		T-A	T-B	T-C	T-D	T-E	T-F
AC-1	Instance	Public	Black (excl: Auto., T-3, VT-8)	Black (excl: North America)	Gray (incl: Europe, excl: IT)	Gray (incl: EU, excl: TC, IT)	Private
	DBMS		Public				
	Virtual Machine			Public	Public	Public	
AC-2	Instance	Public	Black (excl: Auto., T-3)	Black (excl: North America)	Gray (incl: Europe, Asia excl: IT)	Black (excl: TC)	Private
	Application Server		Public				
	Virtual Machine			Public	Public	Public	
AC-3	Instance	Public	Public	Black (excl: North America)	Gray (incl: Europe, Asia excl: IT)	Black (excl: TC)	Private
	Application Server						
	Virtual Machine			Public	Public	Public	
AC-4	Instance	Public	Public	Public	Public	Black (excl: TC)	Private
	Application Server						
	Virtual Machine					Public	
AC-5	Instance	Public	Public	Public	Public	Public	Private
	Physical Server						

Table 4.4.: Deployment Constraints for Scenario 3

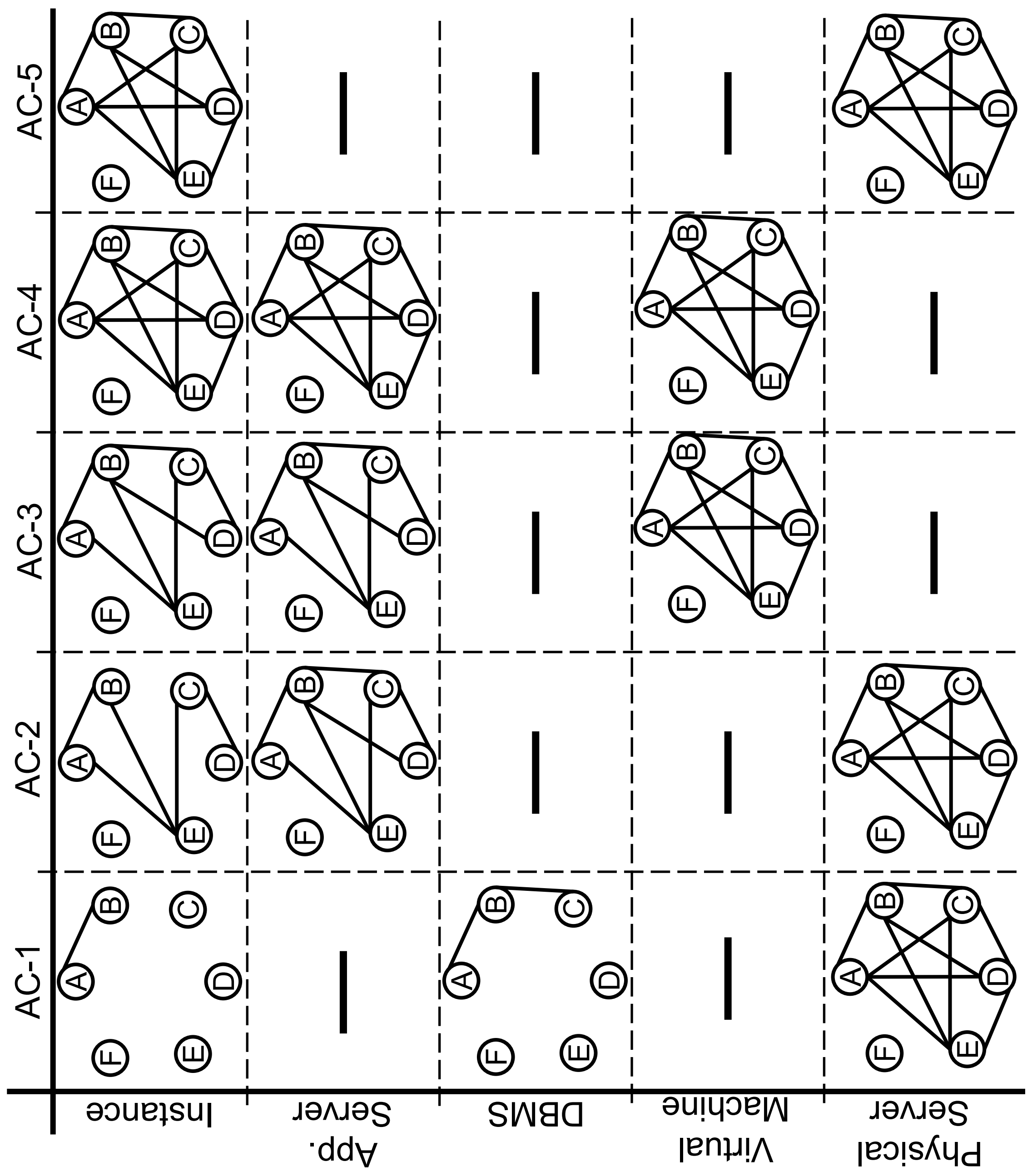

Figure 4.10.: Deployment Information of Scenario 3

4.7 Summary

This chapter addressed the open question of how Deployment Constraints may be captured (research question RQ-1). This was done by defining a Description Model. The objective of this Description Model is to be applicable to a variety of applications and not just for a single application.

This goal was achieved by starting with conducting a requirement analysis. In this analysis it was discussed what the Deployment Model shall be able to capture. The following is an overview.

Deployment Levels It was stated that it shall be possible that Tenants express their Deployment Constraints not just for the Application Components but also for the underlying infrastructure stack.

Deployment Models Five Deployment Models (Private, Public, White Hybrid, Black Hybrid, and Gray Hybrid) were defined. Each offers Tenants another way of expressing with whom they wish to share or not share Deployment Units.

Groups and Dimensions It shall be possible to categorize Tenants according to different topics. Thus, the concept of Groups and Dimensions was introduced.

Virtual Tenants In order to allow Tenants to exclude other Tenants that are not Customers yet, the concept of Virtual Tenants was introduced. This concept allows Operators to keep their Customer base secret.

Based on the requirements analysis, the Utilization Process was introduced that describes how the Deployment Model may be applied for a specific application. Since the objective of the Description Model is to be application-independent, the first step of the process is to customize it to the specific application. Once the Deployment Model has been customized, the necessary information about the Tenants and how they shall be grouped may be captured. This information is then used by the Tenants to describe their Deployment Constraints. After that the Deployment Information is extracted for the deployment computation step.

Based on the requirements analysis and the process, the chapter continued by defining the Description Model. The Deployment Model and its basic idea was introduced superficially by using a UML Class diagram (Figure 4.6). Afterward, the Description Model was defined in all its details using first-order logic.

The formal definition of the Deployment Model was used as a blueprint to create a realization. This realization was done using the semantic web technologies OWL, SPARQL, and SWIRL as well as Java as an implementation language. The realization is an integral part of the Deployment Configuration Generator, which is the tool that was introduced in Section 2.2.5.

The chapter concluded with an evaluation of the Description Model. It was conducted by creating scenarios that consist of all possible cases that may occur based on the requirements analysis. These scenarios were modeled using the realization in OWL to show that the Constraints are describable. The result of the evaluation is that all requirements, defined in the beginning of this chapter, may be described using the realization that was created based on the formal definition of the Description Model. If, however, the requirements defined here are applicable to real applications that may be deployed using the Mixed-Tenancy paradigm, they will be discussed again in a case study presented by Chapter 6.

Considering the results, it can be summarized that this chapter successfully proposed and implemented an approach that serves to capture Customers' Deployment Constraints towards Mixed-Tenancy deployments. The next chapter will discuss how the extracted Deployment Information may be used to calculate a Valid and Optimal Deployment.

Chapter 5
Computation of a Valid and Optimal Deployment

5.1. Fundamental Concepts and Approaches III 89
5.1.1. Additional Concepts of Graph Theory 89
5.1.2. Concepts in Complexity Theory 90
5.1.3. Introduction of Graph Coloring and Clique Cover 91
5.2. Formal Definition of the Deployment Problems 96
5.2.1. Problem Analysis 96
5.2.2. Definition of the General Deployment Problem 100
5.2.3. Definition of the Elementary Deployment Problem . . . 106
5.3. Analysis of Elementary Deployment Problem 109
5.3.1. Definition of a Solution as a Set of Clique Covers 109
5.3.2. Minimal Clique Cover on High Level 110
5.3.3. Minimal Clique Cover on Low Level 113
5.3.4. Minimal Clique Cover on Any Level 116
5.3.5. Complexity of the Elementary Problem 117
5.3.6. Introduction of Heuristics 118
5.4. Analysis of General Deployment Problem 120
5.4.1. Complexity of the General Problem 120
5.4.2. Generalization of Elementary Problem 121
5.4.3. Introduction of Heuristics 121
5.5. Evaluation 124
5.5.1. Experimental Comparison of Heuristics 125
5.5.2. Application on Running Example 127
5.5.3. General Efficiency of Resource Utilization 128
5.6. Summary 131

The computing scientist's main challenge is not to get confused by the complexities of his own making.

Edsger Wybe Dijkstra

In the previous chapter it was analyzed how Customers' Deployment Constraints may be captured using the Description Model. Furthermore, the extraction of Deployment Information was discussed. The Deployment Information is the input based on which a Valid and Optimal Deployment shall be computed. This chapter's purpose is to analyze the challenges involved with performing this computation. By doing so, this chapter contributes to the goals of this work by addressing research question RQ-2. Thus, the following two artifacts are created.

Formal Definition of Optimization Problem The first artifact created in this chapter is a precise and formal definition of the optimization problem. It allows to compare the optimization problem of this work to commonly known problems in literature.

Deployment Computation Algorithm The second artifact is an intuitive algorithm that allows to solve the previously defined optimization problem in a fast but inaccurate way.

> **Definition 24 (Deployment Computation Algorithm)** The *Deployment Computation Algorithm* allows a fast but inaccurate computation of a Deployment that is Valid and strives to be Optimal. This algorithm takes the Deployment Information as input. This means that the computed Deployment is Valid but not Optimal. It is one of the major contributions of this work.

In order to create these artifacts, the chapter is structured as follows. After an introduction of fundamental concepts and approaches required by this chapter (Section 5.1), Section 5.2 starts by discussing and defining the General Mixed-Tenancy Deployment Problem. Furthermore, based on the formal definition of the General Problem, a second problem is defined. This is a simplified version that is referred to as Elementary Mixed-Tenancy Deployment Problem. It is introduced in order to allow to analyze the structure of a Valid and Optimal Deployment in a more readable way. This is done in the next section.

Section 5.3 discusses characteristics of a Valid and Optimal Deployment and determines the complexity of the Elementary Problem. Further, two Deployment Computation Algorithms are presented to tackle the Elementary Problem and their relative performance guarantee is analyzed.

Based on the discussions concerning the Elementary Deployment Problem, Section 5.4 generalizes the gained conclusions to the General Deployment Problem. This includes the characteristics, complexity, and the Deployment Computation Algorithms.

However, in order to also gain some more realistic results, the two Deployment Computation Algorithms will be compared in Section 5.5 by performing an experimental evaluation.

Section 5.6 concludes the chapter with a summary and a conclusion. It will do that by highlighting the contributions of this chapter to the previously defined research questions.

Some of the results presented in this section, especially those related to formalizations and formal proofs, were developed in cooperation with Steffen Lange and Marian Margraf. We also jointly published a paper [LMRV14] that contains some of the results presented in this chapter.

5.1 Fundamental Concepts and Approaches III

Similar to the *Fundamental Concepts and Approaches* of the previous chapters, this section introduces fundamental concepts and approaches relevant for this chapter. In order to do that, additional concepts of graph theory are introduced. In addition, this section gives a brief introduction to complexity theory as this will be relevant for the creation of the Deployment Computation Algorithm and the analysis of the optimization problems. Based on the first two sections, two well-known problems from graph theory will be introduced that are relevant for this work.

5.1.1 Additional Concepts of Graph Theory

In the previous *Fundamental Concepts and Approaches* section (Section 4.1), a first introduction to graph theory has already been given. However, for this chapter additional concepts are relevant. These are standard concepts in graph theory and may, for example, be found in [KN09; Wal07; BR12; Kub04]. They are

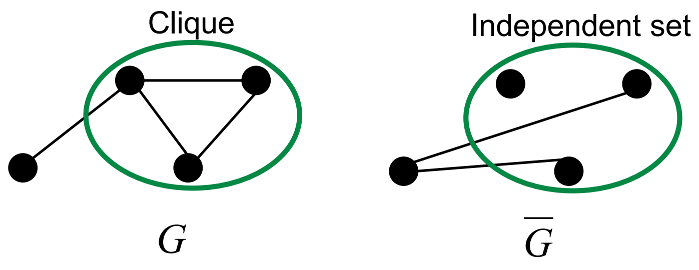

Figure 5.1.: Relationship between Clique and Independent set [KN09, pg. 58]

introduced by the following.

Degree of a Vertex The degree of a vertex is the number of edges it is connected to.

Maximum degree of a Graph The maximum degree of a vertex in a graph is the maximum degree of a graph. The notation to express the maximum degree of graph G is $\Delta(G)$.

Minimum degree of a Graph On the other hand, the minimum degree of a graph is the minimum degree of a vertex in a graph. The notation to express the minimum degree of graph G is $\delta(G)$.

Subgraph A graph G' is a subgraph of graph G if all vertices and edges of graph G' are also in graph G.

Inverse Graph An inverse graph $\overline{G}$ of graph G contains all vertices of G but those vertices adjacent in G will not be adjacent in $\overline{G}$ and those vertices not adjacent in G will be adjacent in $\overline{G}$. Figure 5.1 gives an example.

Clique A clique is a complete subgraph of a given graph. What a complete graph is has been introduced in Section 4.1. Every clique in a given graph G is an independent set in the inverse graph $\overline{G}$.

Independent Set An independent set is a set of vertices belonging to the same graph where no two vertices are adjacent. Figure 5.1 gives an example. Obviously, every independent set in a given graph G is a clique in the inverse graph $\overline{G}$.

Cycle Graph A cycle graph or circular graph is a graph that consists of a single cycle, or in other words, a number of vertices connected in a closed chain. A graph called odd cycle is a cycle graph with an odd number of nodes vertices.

Planar Graph A graph is referred to as planar if there is a drawing of the graph where no edge intersects another edge.

5.1.2 Concepts in Complexity Theory

The following is a brief introduction of concepts known in complexity theory that are relevant for this work. The first relevant concept is the concept of complexity classes.

In computational complexity theory, a complexity class refers to a set of problems that have related resource-based complexity. Many complexity classes are known. For this work the following well established complexity classes are considered relevant [CLRS09; JM08]:

P A decision problem[1] belongs to class P if there exists at least one deterministic Turing machine that solves the problem in polynomial time. This means the number of steps of the algorithm is bounded by the polynomial in n, where n is the length of the input. It is commonly defined that a polynomial time algorithm is fast [Wan06]. This definition is used even though the polynomial execution may be far away from what would informally considered fast (e.g. $O(n^{1000})$).

NP Stands for non-deterministic polynomial time and is the set of all decision problems for which there exists at least one non-deterministic Turing machine that solves the problem in polynomial time.

NP-complete A decision problem is NP-complete if the problem is in NP and every problem in NP is reducible to it in polynomial time.

[1]A decision problem is a problem whose result is either true or false.

NP-hard A decision problem is NP-hard if every problem in NP is reducible to it in polynomial time. An optimization problem is NP-hard if there is an NP-hard (optimization or decision) problem that may be reduced to the NP-hard problem in polynomial time.

The P versus NP problem is one of the major unsolved problems in computer science. It deals with the question of whether all problems in NP are actually problems in P. If this would be the case, fast algorithms exist to solve these problems. The answer is not currently known. It is known that P is a subset of NP. However, it is assumed that P is not NP [JM08].

The second concept relevant for this work is the concept of the relative performance guarantee. For a given optimization problem where the goal is to minimize, the relative performance guarantee (p_A) may be defined as follows.

$$OPT(I) \leq A(I) \leq p_A(|I|) \cdot OPT(I) \tag{5.1}$$

For this definition $OPT(I)$ stands for the quality of an optimal solution for an input I, the length of the input $|I|$, and $A(I)$ for the algorithm used. This means that the relative performance guarantee is the factor according to which a worst case solution is worse in relation to the optimal solution [Wan06].

5.1.3 Introduction of Graph Coloring and Clique Cover

Graph coloring or also called vertex coloring is the problem of segmenting a graph into a number of independent sets of vertices [KN09, pg. 58]. A function $f : V \to 1, 2, \ldots, k$ is called vertex coloring with k colors. If two adjacent vertices (v, v') will be assigned different values ($f(v) \neq f(v')$). Based on this, it may be defined that $\chi\,(G)$ is called the chromatic number of G [JT95].

$$\chi\,(G) = min\{k : \text{there is a coloring of } G \text{ with } k \text{ colors}\} \tag{5.2}$$

It describes the minimum number of colors necessary to color a graph G.

The Graph coloring problem is one of the oldest and best-known problems of graph theory [Kub04]. In fact, the reason why the term coloring is used for this problem originates from coloring the countries of a map [Lon13]. Already in 1852 the observation was documented that coloring the countries on an administrative map of England in a way that adjacent countries were given different colors could be achieved with only four colors [Kub04, pg. x]. A map may be generalized as a graph where the countries are vertices and the nodes determine if two countries are adjacent. In fact, a graph representing a map has a special characteristic: it is planar. After lots of research and some publications that claimed to have found a proof only to be dismissed by others, the first complete proof that all planar graphs may be colored with four colors was presented in [AH77]. In fact, it was the first major theorem proven using a computer [Kub04]. It is commonly referred to as the four color theorem[2]. A major part of the 20th-century research in graph theory actually originates from the four-color problem [BR12].

[2]The claim, sometimes found in literature, that the four color theorem is interesting to cartogra-

However, the graph coloring problem may not just be applied to planar graphs but also to other simple graphs. The problem of coloring has many applications under which the most prominent is register allocation in compilers optimization. Its goal is to assign a large number of target program variables to a small number of CPU registers [CD06].

Graph coloring contains multiple decision problem and an optimization problem. The standard decision problem has two inputs, a graph G and a natural number k. The decision problem is to decide if for there is G a proper vertex coloring with k colors. The complexity of this problem is proven to be NP-complete [Kub04].

Based on this problem, there are other decision problems called k-coloring problems that each deal with a specific k (e.g. 1-coloring, 2-coloring, etc.). It was possible to prove that the complexity of the k-coloring problem is in P for $k \leq 2$ and for all other cases it is NP-complete. The optimization problem, on the other hand, is the problem of finding a coloring that only uses the minimal number of colors. This requires determining the chromatic number. This problem is NP-hard [Kar72]. However, there are entire classes of graphs for which the chromatic number can be calculated through a simple formula or an optimal coloring can be determined in polynomial time [Kub04]. Interval graphs, for example, fall into this category [SVD03].

However, in order to determine a lower bound for the chromatic number, it is easy to observe that a graph that has no edges will be 1-colorable, meaning that it can be colored with only one color. An upper bound, on the other hand, can be given by considering a complete graph that has n vertices. For this graph the chromatic number is n. This leads to the obvious conclusion that the chromatic number of a graph must be at least 1 but cannot be greater than the number of vertices it has.

$$1 \leq \chi(G) \leq n$$

An improvement on the lower bound can be done by considering the number of nodes in the largest clique of a graph (ω). It can be stated that a graph needs at least as many colors as ω.

$$\omega(G) \leq \chi(G)$$

However, the problem with this definition is that there is no known fast algorithm to determine the number of vertices of the largest clique in a graph ($\omega(G)$) since the problem is also NP-hard [Kub04].

For the upper bound it may be proven that the chromatic number is equal or greater than the maximum degree graph G plus one ($\Delta(G) + 1$).

$$\chi(G) \leq \Delta(G) + 1$$

phers, is in fact wrong. This is due to the fact that cartographers see little need to limit how many colors they use. [Wil02]

In fact, it has been proven by [Bro41] that for all graphs, except for complete graphs and odd cycles, the chromatic number is equal or less the maximum degree of a vertex in graph G. Therefore, if G is not a complete graph and $\Delta(G) \geq 3$ than it must be true that:

$$\chi(G) \leq \Delta(G)$$

However, to some classes of graphs, this upper bound is very inaccurate. Such a class is, for example, the class of stars[3]. For those graphs the chromatic number is always 2, while the discussed bound implies $\chi(G) \leq \Delta(G)$ [Kub04; BR12].

However, additional definitions for the upper and lower bound are possible but those presented to this point are sufficient for the purpose of this work.

As stated earlier, the vertex coloring problem has been widely investigated. Further, it has already been stated that its optimization problem is NP-hard. Thus, computing an optimal solution is extremely resource intensive. By solving the k-coloring problem (the decision problem to determine if a graph is colorable with k colors) it is possible to compute the chromatic number through a brute-force algorithm. A graph is tested for every $k = 1, \ldots, n-1$ until a k-coloring is true. The k-coloring can be decided in time $O(2^n n)$ for any k using the approach presented in [BHK09].

Due to the complexity of the problem, it is possible to find a huge number of approximation algorithms in literature. An approximation algorithm is an algorithm that computes a coloring in polynomial time. Due to the complexity of the problem the computed coloring is not optimal. However, an approximation algorithm also states a worst case performance. Furthermore, in [BGS98] it was shown that the problem is not approximateable within $n^{1/7-\epsilon}$ for any $\epsilon > 0$. This means that it is not possible to approximate this problem with a worst case performance quality better than $n^{1/7-\epsilon} \cdot OPT$ were OPT is an optimal coloring (chromatic number) an were n is the number of vertices. The approximation algorithm with the currently best-known performance quality has been presented by [Hal93] and has a quality of $O(n(\log\log n)^2/(\log n)^3)$ [Kub04].

Besides this very sophisticated algorithm, the following is an introduction of two very well-known algorithms.

Largest First The largest first algorithm is one of the oldest methods and has first been introduced by [WP67]. It was created based on the observation that vertices with low degree usually allow for a more flexible choice of colors. Because of that, it is beneficial to assign colors to those vertices with high degree first. In the algorithm this is realized by first sorting all vertices according to the decreasing value of their degree. Based on this list, the algorithm assigns the smallest possible color to each vertex starting with the one with the largest degree and working it way down sequentially.

DSATUR The DSATUR algorithm was first presented by [Bre79]. This algorithm is based on the observation that the constraints in coloring result from

[3]A star is a graph in the shape of a tree with one root and k leaves [BR12].

the number of the uniquely colored neighbors of a vertex. The algorithm performs a sequential coloring with a dynamic established order of vertices [Klo02]. In detail, the algorithm starts with sorting the vertices according to their degree. The vertex with the highest degree will get the first color. The next step is to choose a vertex with a maximal saturation degree. The saturation degree of a vertex is the number of different colors at the vertices adjacent to this vertex [Klo02]. If there should be multiple vertices with the same saturation degree, any uncolored vertex with maximum degree is chosen. The chosen vertex is assigned the lowest possible number. At this point the algorithm continues with, again, choosing a vertex with maximum saturation degree until all vertices are colored.

The DSATUR algorithm colors optimally all bipartite graphs, cycles, mono- and bi-cyclic graphs, trees, necklaces and cacti, as well as all graphs whose core is a member of one of the aforementioned families [Kub04].

Another problem that is well-known in literature, is the clique cover problem. It is known as the problem of determining a minimum number of sets of vertices of a graph, so that all sets are disjoint cliques [Weg03]. This means that there exists a graph $G = (V, E)$. For this graph a set of disjoint cliques CG is to be found so that CG shall only contain a minimum number of cliques c. This is defined by the following.

$$CG = min\{c_1 \sqcup \ldots \sqcup c_n\} \tag{5.3}$$

This problem is categorized to be NP-hard as well [Kar72]. However, in literature, this problem is not covered very extensively. This is due to the fact that [Kar72] reduced vertex coloring to the clique cover problem, which is extensively covered in literature.

Clique cover and vertex coloring are the same problem since the clique cover of a given graph G is, in fact, the same as finding a minimal coloring of the inverse graph $\overline{G}$ [Kar72]. This is due to the fact that vertices that belong to a clique in a given graph are an independent set in the inverse graph. Thus, solving the initial problem of splitting a graph into a disjoint set of cliques may be solved by determining the chromatic number of the inverse graph. Figure 5.2 illustrated the approach by applying it to an example. As visible in the figure, the first step is to create the inverse graph of G. For this graph ($\overline{G}$) the minimum number of independent sets is determined by assigning a color to every vertex. As visible in the figure there are two colors necessary: green and red. One is assinged to the vertices A, B, and C, the other by vertices D and E. Figure 5.2 illustrates one additional step where the graph is reinverted. This shall only illustrate that, in fact, the minimal number of cliques was found. This step is not necessary for the initial problem.

The problem of determining the chromatic number is commonly referred to as vertex coloring. This problem is among Knarp's 21 NP-Complete problems [Kar72]. This means that there is no algorithm to solve this problem in polynomial time unless $P = NP$. However, as already mentioned, this problem has

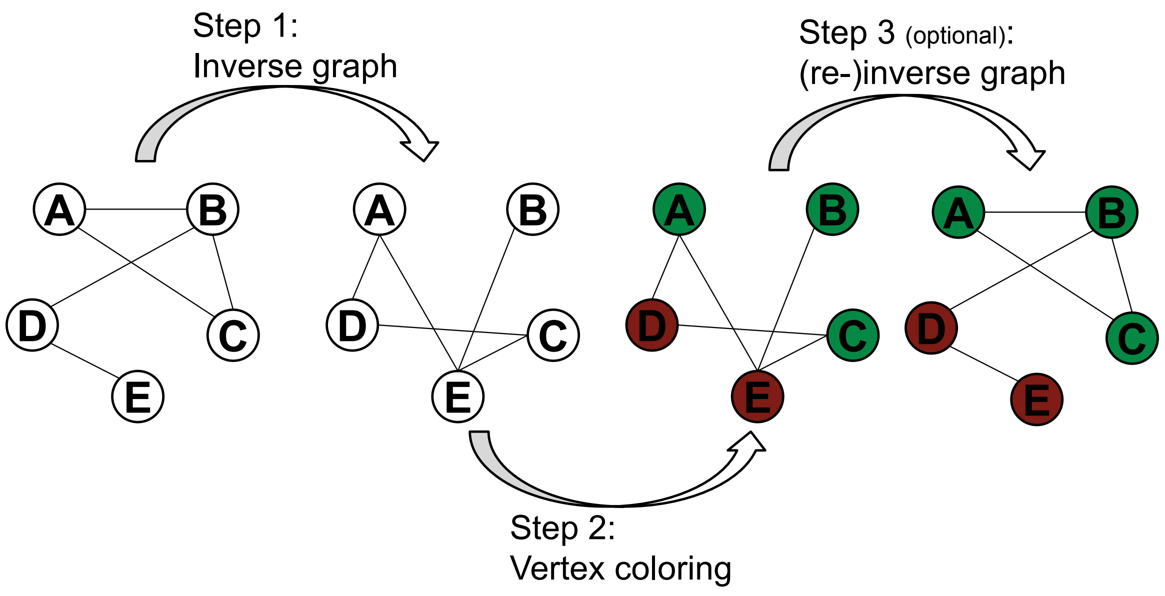

Figure 5.2.: Solving encountered Clique Cover by Coloring the inverse Graph

intensively been researched in the past. Further, there is a high number of known algorithms that try to find solutions with different accurateness in polynomial time and since the two problems are equivalent, all statements made about complexity as well as the lower and upper bounds of the vertex coloring problem apply to the clique cover problem as well.

When dealing with the clique cover problem, this work will utilize the following formal definitions. Let $G = (V, E)$ be an undirected graph. A set $C \subseteq V$ is said to be a clique for G if and only if the subgraph $G_C = (C, E(C))$ induced by C is a complete graph. A family $\mathcal{C}$ of subsets of V is said to be a clique cover for G if the following constraints are met.

$$\forall C \in \mathcal{C} : C \text{ is a clique for } G \tag{5.4}$$

$$\forall C, C' \in \mathcal{C} : C \neq C' \rightarrow C \cap C' = \varnothing \tag{5.5}$$

$$\bigcup_{C \in \mathcal{C}} C = V \tag{5.6}$$

Further, within this work, $M_{cc}(G)$ is the set of all clique covers for G. Finally, let $G = (V, E)$ and $G' = (V', E')$ be undirected graphs with $V = V'$ and $E \subseteq E'$. Moreover, let $\mathcal{C} \in M_{cc}(G)$ and $\mathcal{C}' \in M_{cc}(G')$. The clique cover $\mathcal{C}$ is said to be a refinement of the clique cover $\mathcal{C}'$, denoted by $\mathcal{C} \sqsubseteq \mathcal{C}'$ if the following constraint is met.

$$\forall \mathcal{C} \in M_{cc}(G)\, \forall \mathcal{C}' \in M_{cc}(G') : \mathcal{C} \sqsubseteq \mathcal{C}' \rightarrow \forall C \in \mathcal{C} : \exists C' \in \mathcal{C} \rightarrow C \subseteq C' \tag{5.7}$$

This means that $\mathcal{C}$ is a refinement of the clique cover $\mathcal{C}'$ if and only if for all cliques in $\mathcal{C}$ there exists a C' in $\mathcal{C}$ so that C is a subset of C'.

5.2 Formal Definition of the Deployment Problems

The goal of this section is to properly introduce the optimization problem of this work. This problem it referred to as General Mixed-Tenancy Deployment Problem.

Definition 25 (General Mixed-Tenancy Deployment Problem) It is the problem of finding a Valid and Optimal Deployment. It is the primary optimization problem that is defined as part of this work.

In order to define this problem, this section will start with a problem analysis (Subsection 5.2.1) where the objectives of the optimization problem will be introduced and discussed. Based on this, Subsection 5.2.2 defines the General Mixed-Tenancy Deployment Problem formally.

Furthermore, Subsection 5.2.3 introduces a second problem. This problem is a simplified version of the General Problem and, thus, it is referred to as Elementary Mixed-Tenancy Deployment Problem.

Definition 26 (Elementary Mixed-Tenancy Deployment Problem) The *Elementary Mixed-Tenancy Deployment Problem* is a simplified version of the Mixed-Tenancy Deployment Problem. The only difference between both problems is that in the Elementary version there is only one Application Component.

It is introduced in order to allow to analyze the structure of a Valid and Optimal Deployment in a more readable way in the next section.

5.2.1 Problem Analysis

In Section 2.2.3 it was defined that the general research question of this work is to find a way how an Operator may still have minimal cost for IT-infrastructure, even if Customers' Deployment Constraints need to be considered. In the previous chapter it was discussed how those Constraints may be captured and the Deployment Information is extracted. The next step discussed by this chapter is how a Valid and Optimal Deployment can be computed.

According to the research questions of this work, a Valid and Optimal Deployment shall apply to all Deployment Constraints but still cause as little cost as possible for the Operator. This was stated in the general research question of this work as well as in research question RQ-2 and the definition of the Valid and Optimal Deployment. However, before it may be analyzed how the deployment's cost can be optimized, it must first be discussed what the Valid Deployment shall look like in more detail.

So far it has always been stated that a Valid Deployment shall apply to the Deployment Constraints expressed by the Customers. Thus, this can be expressed as the characteristic for a Valid Deployment.

Customer Constraints are Considered A Deployment is only Valid if it complies with all Deployment Constraints given by Customers.

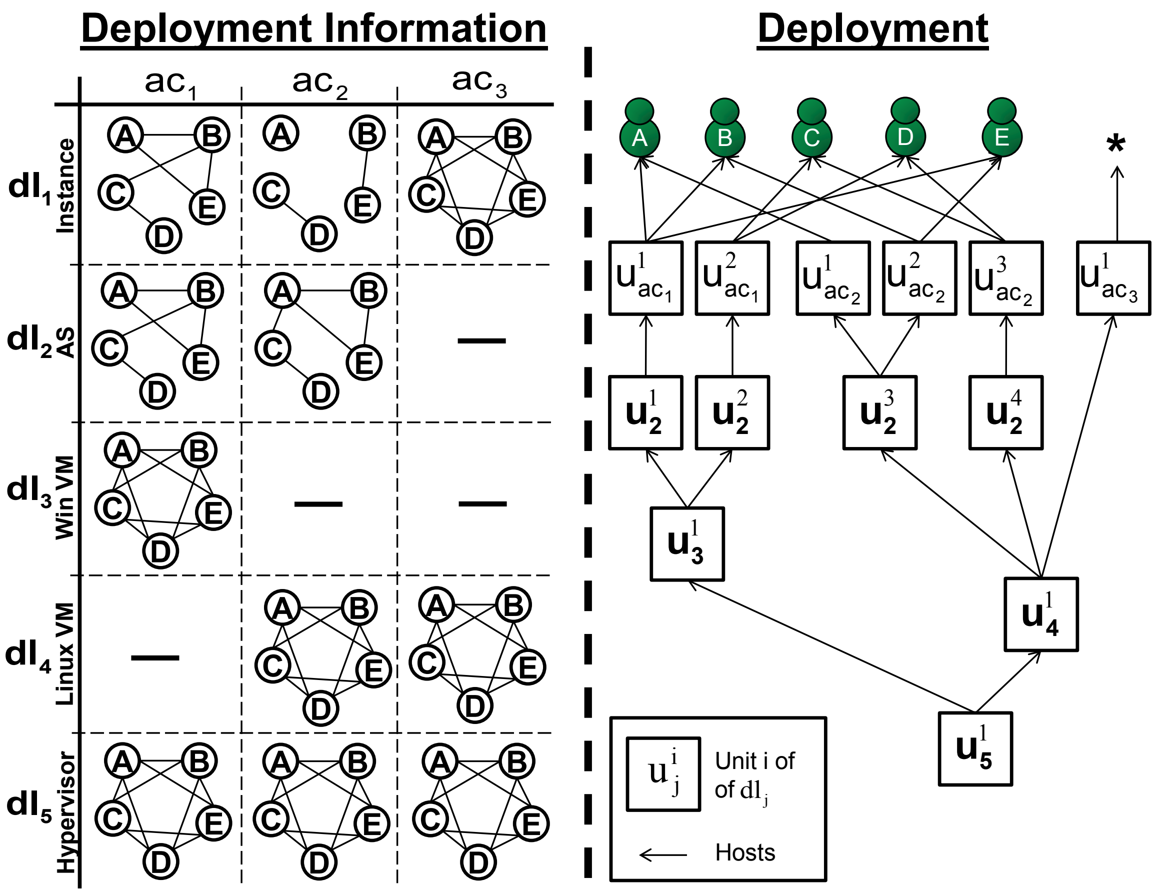

Figure 5.3.: An Example of a Valid and Optimal Deployment

Figure 5.3 illustrates this based on an example. In this example five Tenants are using an application that is composed of three Application Components. The input is the Deployment Information visualized as graphs (as it has been discussed by Sections 4.3.4 and 4.4.8). Furthermore, the Application Components are deployed on five Deployment Levels, where not all are used by all Application Components. One constraint that was defined concerning the relationship between Application Components and Deployment Levels is that all Application Components must be deployed on the first Level dl_1. This is due to the fact that this Level represents the instantiation of Application Components. As this work focuses on the deployment of Multi-Tenancy applications, all Application Components must be instantiated. In the example (Figure 5.3) the Deployment Levels stand for the following: dl_1 - Instance, dl_2 - application server hosting Application Components, dl_3 - virtual machine running Windows, dl_4 - virtual machine running Linux, and dl_5 - hypervisor able to run different virtual machines. Further, it is visible in the figure that not all Application Components are deployed on all Levels. Application Component ac_1 requires an application server (dl_2) and a virtual machine running Windows (dl_3). This may, for example, be due to the fact that the Application Component requires some special resource only being available for Windows. The Application Component ac_2 and ac_3 both require a Linux virtual machine (dl_4) but only ac_2 requires an application server (dl_2).

On the left hand side of Figure 5.3 it is also visible how Customers are willing to share Units in the example. For Application Component ac_1 and Level dl_1, for example, Tenants A and B would be allowed to share a Unit, Tenant A and C, on the other hand, would not be allowed to share. Further, it can be said that ac_3 must be a Application Component that handles only insensitive data, as all Tenants are willing to share this Application Component on all Deployment Levels. It is important to note that the graphs belonging to the same Application Component must always get less or equally restrictive from top to bottom (this has previously been stated in Section 4.4.8).

A Valid and Optimal Deployment for the example is illustrated on the right hand side of Figure 5.3. As visible for every Deployment Level, Units were created. In fact, on Level dl_1 there are Units for every Application Component. On the other Levels there is only one type of Unit per Level. Tenants are directly connected to those dl_1 Level Units they use. Each Tenant uses one Unit of each Application Component. All dl_1 Units are hosted on Units of the lower Levels. This structure directly corresponds to the structure of Application Components and the Deployment Levels they are deployed on. Thus, the Units u_2^3, u_2^4, and $u_{ac_3}^1$ are deployed on a Unit of Level dl_4 but not on a Unit of Level dl_3.

Now that it has been established what a Valid Deployment shall look like, it is possible to analyze how cost of a Deployment may be optimized. In order to determine what factors actually have an impact on the cost of a deployment, it is necessary to take another look to what a Deployment looks like and analyze what actually causes cost. In general, the cost of a Deployment is determined by the demand for resources as this is what causes cost.

Definition 27 (Resource) The term *Resource* is used as a generic term for all kinds of possible IT resources. The most obvious are CPU power, RAM size, and Disk space. However, in this work the term resources refers to everything that is consumed by a Unit of a Deployment Level and has a potentially infinite availability.

It is possible to distinguish two types of resource demands, those caused by the Tenants and those caused by the Units.

Definition 28 (Resource Demand of Tenant) By using the application provided by the Operator, each Tenant causes a resource demand. What the resource demand for a Tenant is depends on many factors, as for example the number of Users that shall use the application, or how intensively these Users are using it. In fact, a Tenant's resource demand may vary annually, monthly, or even depending on the time of the day.

Definition 29 (Resource Demand of Unit) The second resource demand is the one caused by each Unit itself. It is independent from how many Tenants are using the application, it only refers to the overhead cost that comes with provisioning a Unit. The Resource Demand of Units vary, as they may be instances of different Deployment Levels.

In order to elaborate further on how infrastructure cost may actually be optimized, Figure 5.4 illustrates a theoretical experiment. In this experiment the same n Tenants are set up to either use 1 or n Units. In both cases, the Tenants are the same and so is their resource demand. Furthermore, in both cases there exists only one type of Unit. For both cases the total resource demand (TRD) may be

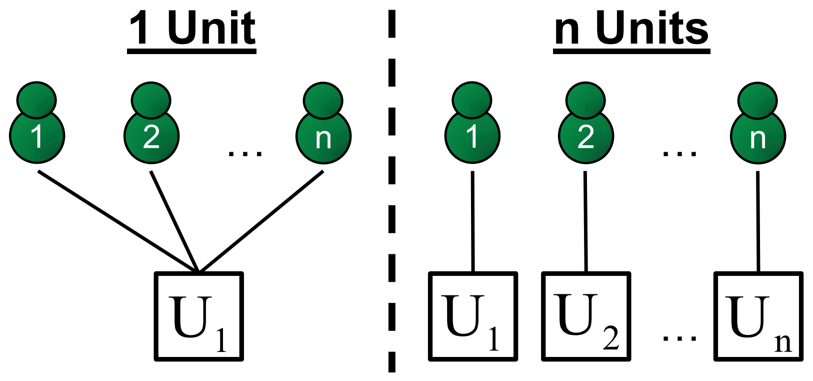

Figure 5.4.: Extremes how Customers may share Units of one Deployment Level

calculated by adding the Resource Demand caused by Tenants (RC_T) and the Resource Demand caused by the Units (RC_U).

$$TRD_{1\text{ Unit}} = \sum_{i=1}^{n} RD_{T_i} + RD_{U_1}$$
$$TRD_{\text{n Units}} = \sum_{i=1}^{n} RD_{T_i} + \sum_{j=1}^{n} RD_{U_j}$$

As visible in the equations, the difference between $TRD_{1\text{ Unit}}$ and $TRD_{\text{n Units}}$ is the Resource Demand caused by the Units. Since the Resource Demand caused by the Tenants may not be altered, this leads to the conclusion that the only way how infrastructure cost may be optimized is by optimizing the cost caused by Units. Thus, this is the starting point for the formal definition of the General Mixed-Tenancy Deployment Problem.

This work is conducted under the assumption that the Resource Demand of Tenants does not need to be considered when calculating a Valid and Optimal Deployment. This is due to the fact that in recent years there has been significant advances in the area of resource virtualization that allows to scale resources according the demands up and down [Feh+13]. Thus, it is an assumption of this work that any Unit may run any number of higher Level Units, without having any restrictions to resource availability. However, there may be cases where this assumption may be false. An example for this is a Software License that allows only a specific number of users per software installation. These cases should be avoided in the area of Mixed-Tenancy as they increase complexity of computation significantly. However, this problem is out of scope for this work. Anyway, it will be discussed briefly in Appendix C.

5.2.2 Definition of the General Deployment Problem

Based on the analysis of Section 5.2.1, it is now possible to define the General Mixed-Tenancy Deployment Problem formally. The goal of this section is to do that in a complete, unambiguous, and formal way. The formalization will be used in future sections to analyze characteristics of a Valid and Optimal Deployment and to formulate possible algorithms tackling it. This definition is the first artifact to be created by this chapter.

In order to be able to formulate a lean and understandable formalization, a representation of a Deployment is used that is slightly different from the one used in the previous section. This representation is illustrated by Figure 5.5. The only difference between this representation and the one of the previous

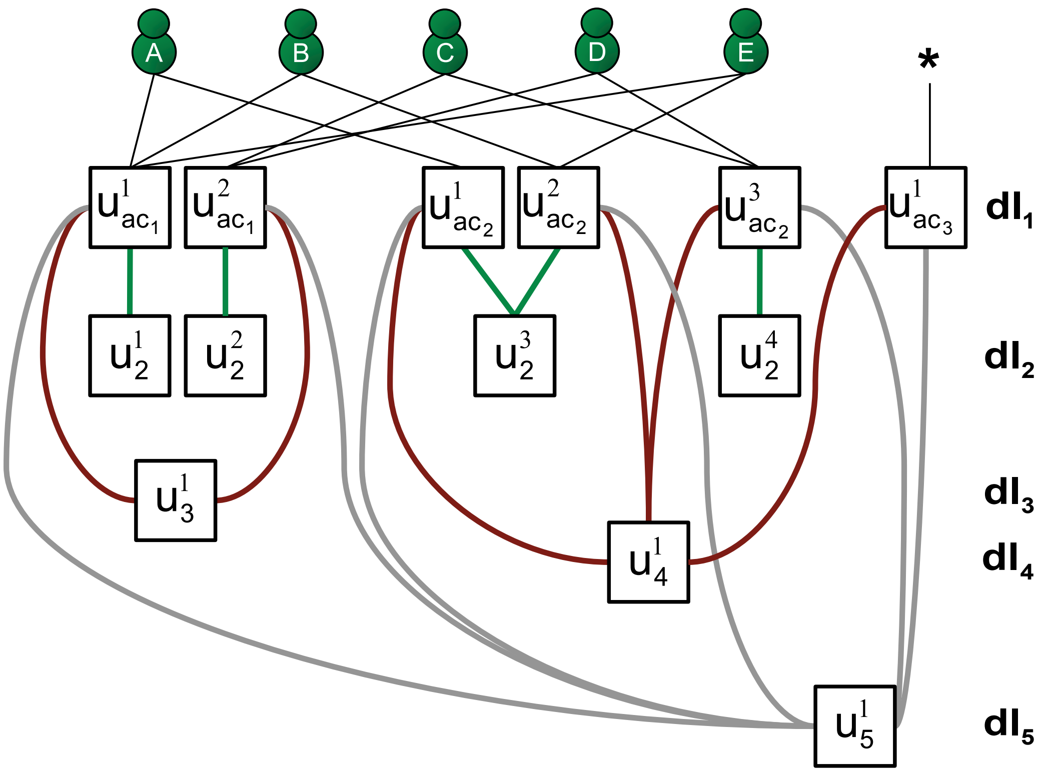

Figure 5.5.: Description of Example's Solution according to formal Definition

section lies in the associations between Units. Instead of associating each Unit to the Units it hosts, they are associated to Units of Level dl_1 they host. Once the formalization of the General Problem has been created, it will be obvious that the representation of the previous section and the one of this section are, in fact, equivalent.

Definition of Inputs

The first step towards the definition of the General Mixed-Tenancy Deployment Problem is the definition of the relevant inputs that are given by previous steps. The first three are, a finite set of Tenants, a finite set of Application Components, and a finite set of Deployment Levels. These are captured using the Description Model. Their formal definitions were given in Section 4.3 and are only posted here again to increase readability.

$$T = \{t_1, \ldots, t_{it}\} \quad : \quad \text{Tenants} \qquad \text{(copy of 4.1)}$$
$$AC = \{ac_1, \ldots, ac_{iac}\} \quad : \quad \text{Application Components} \qquad \text{(copy of 4.2)}$$
$$DL = \{dl_1, \ldots, dl_{idl}\} \quad : \quad \text{Deployment Levels} \qquad \text{(copy of 4.3)}$$

If applied to the example of the previous section, the three sets look as follows.

$$T = \{A, B, C, D, E\}, \quad AC = \{ac_1, ac_2, ac_3\}, \quad DL = \{dl_1, dl_2, dl_3, dl_4, dl_5\}$$

Furthermore, the Description Model, presented in Section 4.4), captured which Application Components shall be deployed on which Deployment Levels. It was defined in Section 4.4 that each Application Component may be deployed on one to many Deployment Levels, but not necessarily on all of them[4]. This leads to the conclusion that every Deployment Level actually has a number of Application Components that it applies to. In Section 4.4.8 the family $\mathcal{DLS}$ was introduced to be another alternative representation of the Deployment Levels, which were represented as sets of Application Components. All DLS were defined to be a set of Application Components that are deployed on a particular Deployment Level.

$$\mathcal{DLS} = \{DLS_{dl_1} \mid DLS_{dl_1} = AC\} \cup \{DLS_{dl_i} \subseteq AC \mid 2 \leq i \leq idl\} \qquad \text{(copy of 4.50)}$$

When applying this to the example, introduced in Subsection 5.2.1, the result looks as follows.

$$\mathcal{DLS} = \{DLS_{dl_1}, DLS_{dl_2}, DLS_{dl_3}, DLS_{dl_4}, DLS_{dl_5}\}$$

$$DLS_{dl_1} = \{ac_1, ac_2, ac_3\}, \quad DLS_{dl_2} = \{ac_1, ac_2\}, \quad DLS_{dl_3} = \{ac_1\},$$
$$DLS_{dl_4} = \{ac_2, ac_3\}, \quad DLS_{dl_5} = \{ac_1, ac_2, ac_3\}$$

The next part of the input, that is necessary to be defined, is the Deployment Information. It is the major output of the previous chapter and describes which Tenants may share Units. The description is given for every Application Component on every Deployment Level it is deployed on. This has previously (Section 4.4.8) been defined as a set $\mathcal{DI}$ of graphs.

$$\mathcal{DI} = \{DI_{dl}^{ac} = (V_{dl}^{ac}, E_{dl}^{ac}) \mid dl \in DL \land ac \in DLS_{dl}\} \qquad \text{(copy of 4.52)}$$

This means that if two Tenants are in a set DI_{dl}^{ac}, those two Tenants are allowed to share the combination of Application Component ac and Deployment Level dl. Furthermore, it was defined in Sections 4.3.3 and 4.4.8 that lower Level Deployment Constraints are equally or less restrictive than their predecessors.

[4]This is true for the Deployment Levels dl_2– dl_{idl}. It was defined that all Application Components must be deployed on Deployment Level dl_1

This also still applies. For the problem definition this was defined as follows.

$$\forall dl, dl' \in DL \quad \forall ac \in DLS_{dl} \cap DLS_{dl'} : dl < dl' \rightarrow E^{ac}_{dl} \subseteq E^{ac}_{dl'} \qquad \text{(copy of 4.56)}$$

Furthermore, Section 4.4.8 defined additional constraints for the Deployment Information that must be met. These are not repeated at this point.

In order to elaborate the meaning of the definitions further, the following illustrates the definition of the Deployment Information for the example. Since there are many Deployment Constraints, the following is limited to the set of edges contained by the Deployment Information of Level dl_1 and Application Component ac_1.

$$E^{ac_1}_{dl_1} = \{(A, B), (A, E), (B, E), (B, C), (C, D)\}$$

It was stated in the previous section that the optimization criteria is to find a Deployment that causes only minimal cost. Further, it was stated that the only cost that needs to be considered is the cost that is caused by the Resource Demand caused by Units. Thus, the final piece of information that is necessary to define as input is the cost of each Unit. This input has not been presented so far. However, it is done in the following by defining two functions that assign a cost to every Application Component and Deployment Level.

$$w_{AC} : AC \rightarrow \mathbb{R}^{>0} \tag{5.8}$$
$$w_{DL} : DL \setminus \{dl_1\} \rightarrow \mathbb{R}^{>0} \tag{5.9}$$

According to these definitions the functions allow the following assignments.

- w_{AC} assigns the cost of a Unit of $ac_1, \ldots, ac_{iac}$.
- w_{DL} assigns the cost of a Unit of $dl_2, \ldots, dl_{idl}$.

As visible in Equations 5.8 and 5.9 the Deployment Levels dl_1 and $dl_2, \ldots, dl_{idl}$ are treated differently. For Deployment Level dl_1 the cost of Units must be given for every Application Component. For the Deployment Levels $dl_2, \ldots, dl_{idl}$, on the other hand, cost of Units is only expressed per Deployment Level. This is due to the fact that for Deployment Level dl_1 it is necessary to create Units per AC. Those Units may then be deployed on the Units of the dl_2.

Based on the previous discussion, it can be concluded that the Mixed-Tenancy Deployment Problem requires the following input $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$. Based on this definition, it is possible to continue by defining a Valid Deployment.

Definition of a Valid Deployment

Based on the definition of inputs, provided by the previous subsection, the definition of a Valid Deployment can be given. The first step towards this is to characterize what it means that a Unit of a particular Deployment Level is allowed to host dl_1 Units that are instances of different Application Components. Due to the fact that not every Application Component is necessarily deployed on all Deployment Levels, it is not possible to deploy any two dl_1 Units together on the same lower Unit. The example of Section 5.2.1 illustrates why this is necessary. If it would be possible to deploy any dl_1 Unit with any other dl_1 Unit, it would be possible to distribute the Units of dl_1 onto Units of dl_2 in the way that both instances of ac_1 and ac_2 may share a Unit of dl_2. The result is illustrated by Figure 5.6. The Units created in this way would not be deployable on the Levels dl_2 and

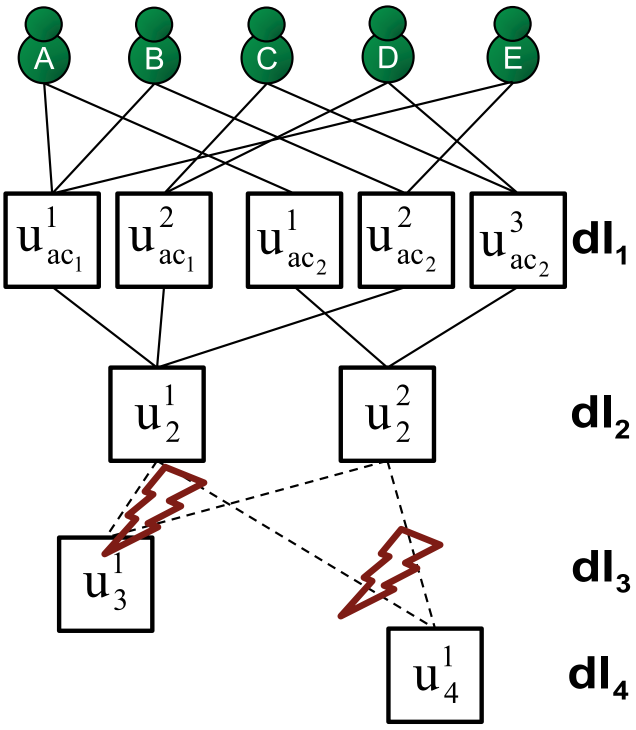

Figure 5.6.: Example why $\sim_{dl_i}$ is Required

dl_3 since they are only deploying either ac_1 or ac_2. This is also indicated by the figure.

Preventing such problems can be accomplished by introducing restrictions specifying which Application Components may share Units for every Deployment Level. Due to the fact that dl_1 Units are created for each Application Component individually, the definition of the Application Component restrictions only needs to be defined for the Deployment Levels dl_2 - dl_{idl}. This is done in the following by defining an equivalence relation ($\sim_{dl_i}$) that determines if dl_1 Units deploying two Application Components may share a Unit on a given Deployment Level.

$$\begin{aligned}\forall dl_i \in DL \quad \forall ac, ac' \in DLS_{dl_i} : i > 1 \wedge ac \sim_{dl_i} ac' \rightarrow \forall dl_j \in DL : j > i \\ \rightarrow \{ac, ac'\} \subseteq DLS_{dl_j} \vee \{ac, ac'\} \cap DLS_{dl_j} = \varnothing\end{aligned} \tag{5.10}$$

This means that two Application Components are allowed to share a Unit on a given Level if and only if they are also deployed on the same Deployment Levels below that. This also means that there is no lower Deployment Level where only one of the Application Components is deployed. Using this definition and applying it to the example will result in the following equivalence classes.

$$\begin{aligned}
&[ac_1]_{\sim_{dl_2}} = \{ac_1\}, && [ac_2]_{\sim_{dl_2}} = \{ac_2\}, && [ac_3]_{\sim_{dl_2}} = \{ac_3\}, \\
&[ac_1]_{\sim_{dl_3}} = \{ac_1\}, && && \\
&[ac_2]_{\sim_{dl_4}} = \{ac_2, ac_3\}, && [ac_3]_{\sim_{dl_4}} = \{ac_2, ac_3\}, && \\
&[ac_1]_{\sim_{dl_5}} = \{ac_1, ac_2, ac_3\}, && [ac_2]_{\sim_{dl_5}} = \{ac_1, ac_2, ac_3\}, && [ac_3]_{\sim_{dl_5}} = \{ac_1, ac_2, ac_3\}
\end{aligned}$$

Using the equivalence relation, it is now possible to define a Valid Deployment for a General Mixed-Tenancy Deployment Problem $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$ as a tuple containing a family of Units and a family of two-place functions.

$$D = (\mathcal{U}, \mathcal{F}) \tag{5.11}$$

$$\mathcal{U} = \{U_{dl_1}, \ldots, U_{dl_{idl}}\} \tag{5.12}$$

$$\mathcal{F} = \{f_{dl_1}, \ldots, f_{dl_{idl}}\} \tag{5.13}$$

For this solution the following variables are defined as follows.

$U_{dl_1}, \ldots, U_{dl_{idl}}$ each represents a set that contains all Units for the corresponding Deployment Level.

$f_{dl_1}, \ldots, f_{dl_{idl}}$ each represents a two-place function that assigns to every $t \in T$ and every $ac \in AC$ a Unit of the corresponding Deployment Level.

The first step towards defining the constraints for a Valid Deployment is the definition of the functions $f_{dl_1}, \ldots, f_{dl_{idl}}$.

$$f_{dl_1} : T \times AC \to U_{dl_1} \tag{5.14}$$

$$\vdots$$

$$f_{dl_{idl}} : T \times AC \to U_{dl_{idl}} \tag{5.15}$$

This defines that each function assigns a given Tenant and a given Application Component to a Unit on the concerning Deployment Level. In order to define the constraints of how Tenants are assigned to Units, it is necessary to distinguish two cases. This is due to the fact that on Deployment Level dl_1 Units are created per Application Component. On the other Level Units only a single type of Unit is created per Deployment Level.

It was discussed earlier that on Deployment Level dl_1 two Tenants shall only share a Unit of a given Application Component if they are allowed to according to the Deployment Information. Further, it has been stated earlier that Units on Deployment Level dl_1 shall only be allowed to host one type of Application Component. Both constraints are expressed by the following equation.

$$\begin{aligned} \forall t, t' \in T \quad \forall ac, ac' \in DLS_{dl_1} : f_{dl_1}(t, ac) = f_{dl_1}(t', ac') \\ \to ac = ac' \wedge (t, t') \in E^{ac}_{dl_1} \end{aligned} \tag{5.16}$$

If this definition is applied to the example of Subsection 5.2.1, the following is the result for Application Component ac_1.

$$f_{dl_1}(A, ac_1) = u^1_{ac_1}, \quad f_{dl_1}(B, ac_1) = u^1_{ac_1}, \quad f_{dl_1}(C, ac_1) = u^2_{ac_1},$$
$$f_{dl_1}(D, ac_1) = u^2_{ac_1}, \quad f_{dl_1}(E, ac_1) = u^1_{ac_1}$$

For the Deployment Levels dl_2 – dl_{idl} only a single type of Unit is created per Deployment Level. It has been discussed earlier that on these Levels, two Tenants shall only be allowed to share a Unit if they are allowed to do so according to the Deployment Information. Since there is only one type of Unit to be created, it is not sufficient to just check the Deployment Information related to one Application Component. In fact, the Deployment Information related to all Application Components involved need to be checked. Further, it has been discussed earlier that not all Instances of Application Components shall be allowed to share Units on all Levels. In order to be able to express this, an equivalence relation was introduced. The following is a constraint that captures all this.

$$\begin{aligned} &\forall dl \in DL \setminus \{dl_1\} \quad \forall t, t' \in T \quad \forall ac, ac' \in DLS_{dl} : f_{dl}(t, ac) = f_{dl}(t', ac') \\ &\quad \rightarrow ac \sim_{dl} ac' \wedge (t, t') \in E^{ac}_{dl} \wedge (t, t') \in E^{ac'}_{dl} \end{aligned} \tag{5.17}$$

In order to be able to define what a Valid Deployment looks like, there is only one constraint left to be defined. This constraint shall express that if two Tenants are assigned to share a Unit on a high Level, they shall also be assigned to Units on lower Levels together. This is due to the fact that if two Tenants are assigned to share the same instance of an Application Component, it is not possible for them to not share the same virtual machine this instance is deployed on. Thus, the next constraint expresses just that.

$$\begin{aligned} &\forall t, t' \in T \quad \forall dl_i, dl_j \in DL \quad \forall ac, ac' \in DLS_{dl_i} \cap DLS_{dl_j} : \\ &\qquad i < j \wedge f_{dl_i}(t, ac) = f_{dl_i}(t', ac') \rightarrow f_{dl_j}(t, ac) = f_{dl_j}(t', ac') \end{aligned} \tag{5.18}$$

Due to this definition, it is possible to say that the representation of a Deployment used in the formal definition (Figure 5.5) and the one of Section 5.3 (Figure 5.3) are equivalent.

If all this is applied to the example of the previous section, the following results may be produced for all Units used by Tenant A.

$$f_{dl_2}(A, ac_1) = u^1_2, \quad f_{dl_3}(A, ac_1) = u^1_3, \quad f_{dl_5}(A, ac_1) = u^1_5,$$
$$f_{dl_2}(A, ac_2) = u^3_2, \quad f_{dl_4}(A, ac_2) = u^1_4, \quad f_{dl_5}(A, ac_2) = u^1_5,$$
$$f_{dl_4}(A, ac_3) = u^1_4, \quad f_{dl_5}(A, ac_3) = u^1_5$$

Definition of a Valid and Optimal Deployment

In the previous subsection it was defined what a Valid Deployment shall look like. Based on this, it is possible to define the cost a Deployment causes by providing a cost function. A first step towards this is to define functions that allow to determine the cost of an entire Deployment Level. Again, it is necessary to treat the Deployment Level dl_1 differently from the others, as Units are created for every Application Component. This is expressed by the following definitions.

$$w(U_{dl_1}) = \Sigma_{ac \in AC}\, w_{AC}(ac) \cdot |\{f_{dl_1}(t, ac) | t \in T\}| \tag{5.19}$$

$$\forall dl \in DL \setminus \{dl_1\} : w(U_{dl}) = w_{DL}(dl) \cdot |U_{dl}| \tag{5.20}$$

Based on this function, it is now possible to express the cost function of a complete Deployment.

$$w((\mathcal{U}, \mathcal{F})) = \Sigma_{dl \in DL}\, w(U_{dl}) \tag{5.21}$$

As defined the cost of a Deployment is the sum of all cost caused by the Units it utilizes. Based on this, it is now possible to formulate the optimization problem of this work as follows: Given any General Mixed-Tenancy Deployment Problem $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$, determine a Valid Deployment that causes minimal cost $w((\mathcal{U}, \mathcal{F}))$.

5.2.3 Definition of the Elementary Deployment Problem

In order to be able to analyze characteristics of a Valid and Optimal Deployment, it is convenient to define a simplified version of the optimization problem. This problem is referred to as Elementary Mixed-Tenancy Deployment Problem.

The only difference between the General and the Elementary Problem is that there exists only one Application Component in the Elementary version. In reality, the Elementary Mixed-Tenancy Deployment Problem may, for example, be found if an Operator chooses to offer an Application to Customers that cannot be split into multiple Application Components. In such a scenario there would only be one Application Component which is, in fact the, application itself.

For the formalization of the problem there are two implications coming from the limitation that there shall only be one Application Component. These are the following.

No unused Deployment Levels In the definitions given in Chapter 4 (Equation 4.20) it was defined that for every Deployment Level there must at least be one Application Component that is deployed on it. In the Elementary Problem there is only one Application Component. This Application Component is deployed on all Deployment Levels.

All Units are Created Equally In the General version of the Deployment Problem, it was necessary to distinguish how Units are created on dl_1 and the other Deployment Levels. Since there is only one Application Component, it is not necessary to make that difference anymore. Thus, for the Elementary Problem, Units are created the same way for every Deployment Level.

The following is a formal definition of the Elementary Mixed-Tenancy Deployment Problem.

Definition of Input

In order to define the Elementary Mixed-Tenancy Deployment Problem, a first step is the definition of inputs. These are quite similar but not the same as the input for the General Mixed-Tenancy Deployment Problem. The first two necessary inputs are a set of Tenants and a set of Deployment Levels. These two may actually be taken directly from the definitions of Chapter 4.

$$T = \{t_1, \dots, t_{it}\} \quad : \quad \text{Tenants} \qquad \text{(copy of 4.1)}$$
$$DL = \{dl_1, \dots, dl_{idl}\} \quad : \quad \text{Deployment Levels} \qquad \text{(copy of 4.3)}$$

Further, it is necessary to have the Deployment Information as input. In the General Mixed-Tenancy Deployment Problem, the Deployment Information was given per Deployment Level and Application Component. Since there is only one Application Component in the Elementary Deployment Problem, it is only necessary to have the Deployment Information once per Deployment Level. Otherwise the definition is the same as the one given for the General Deployment Problem (Subsection 4.4.8).

$$\mathcal{DI} = \{DI_{dl} = (V_{dl}, E_{dl}) \mid dl \in DL\} \tag{5.22}$$
$$\forall dl \in DL : V_{dl} = T \tag{5.23}$$
$$\forall dl \in DL : E_{dl} \subseteq T \times T \tag{5.24}$$

Again, the set of edges needs to be defined as reflexive and symmetric.

$$\forall dl \in DL \quad \forall t \in T : (t, t) \in E_{dl} \tag{5.25}$$
$$\forall dl \in DL \quad \forall t, t' \in E_{dl} : (t, t') \in E_{dl} \tag{5.26}$$

Similar to the General Deployment Problem, it is necessary to define that if two Tenants may share Units of a high Level, they shall also be allowed to share Units on lower Levels. Again, this is expressed by a subset definition on the Deployment Information.

$$\forall dl, dl' \in DL : dl < dl' \rightarrow E_{dl} \subseteq E_{dl'} \tag{5.27}$$

The only input that is left to be defined is the cost functions that allows to determine the cost of every Unit. In the General version of the Deployment Problem it was necessary to define multiple cost functions – one for Deployment Level dl_1 assigning the cost for every Application Component, and one for the other Deployment Levels assigning cost per Deployment Level. However, since in the Elementary version of the Deployment Problem there is only one Application Component, it is sufficient to only define one cost function that assigns cost per Deployment Level. This is done in the following.

$$w : DL \rightarrow \mathbb{R}^{>0} \tag{5.28}$$

Based on this discussion, it can be concluded that a given Elementary Mixed-Tenancy Deployment Problem requires the following inputs $(T, DL, \mathcal{DI}, w)$.

Definition of Valid Deployment

Based on the definition of inputs, it is now possible to define what a Valid Deployment shall look like. Again, this is quite similar to the definition of the normal Deployment Problem.

A Deployment for an Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$ shall be defined as a tuple containing a family of Units and a family of functions.

$$D = (\mathcal{U}, \mathcal{F}) \tag{5.29}$$

$$\mathcal{U} = \{U_{dl_1}, \ldots, U_{dl_{idl}}\} \tag{5.30}$$

$$\mathcal{F} = \{f_{dl_1}, \ldots, f_{dl_{idl}}\} \tag{5.31}$$

Similar to the definition of the General Mixed-Tenancy Deployment Problem, each $U_{dl_1}, \ldots, U_{dl_{idl}}$ contains a set of Units for the Deployment Level. The functions, on the other hand, are not two-placed functions this time, but are only simple functions assigning a Unit to every given Tenant.

$$f_{dl_1} : T \to U_{dl_1} \tag{5.32}$$

$$\vdots \tag{5.33}$$

$$f_{dl_{idl}} : T \to U_{dl_{idl}} \tag{5.34}$$

Two Tenants shall only be assigned to share a Unit on a given Deployment Level if they are allowed to do so according to the Deployment Information. Further, if two Tenants are assigned the same Unit on a given Level, they must also be assigned to the same Units on lower Levels. These two constraints are expressed by the following equations.

$$\forall dl \in DL \quad \forall t, t' \in T : f_{dl}(t) = f_{dl}(t') \to (t, t') \in E_{dl} \tag{5.35}$$

$$\forall dl, dl' \in DL \quad \forall t, t' \in T : dl < dl' \wedge f_{dl}(t) = f_{dl}(t') \to f_{dl'}(t) = f_{dl'}(t') \tag{5.36}$$

Based on these definitions, it is clear what a Valid Deployment shall look like.

Definition of Valid and Optimal Deployment

In order to be able to express what a Valid and Optimal Deployment shall look like, it is necessary to define a cost function, assigning cost to an entire Deployment. Similar to the definition of the normal Deployment Problem, the cost of a Deployment shall be the cost of all Units required by it. This is expressed by the following.

$$w((\mathcal{U}, \mathcal{F})) = \Sigma_{dl \in DL}\, w(dl) \cdot |U_{dl}| \tag{5.37}$$

Based on this, the Elementary optimization problem can be formulated as follows. Given any Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$, determine a Valid Deployment that causes minimal cost $w((\mathcal{U}, \mathcal{F}))$.

5.3 Analysis of Elementary Deployment Problem

In the previous section the Elementary Mixed-Tenancy Deployment Problem has been defined in order to serve as a less complex problem to be analyzed. Thus, it is used in this section in order to analyze characteristics of a Valid and Optimal Deployment and to determine its complexity.

In order to do that the section is structured as follows. The first step taken is to present an alternative representation of a Valid and Optimal Deployment based on concepts from graph theory (Subsection 5.3.1). Based on this definition, the Subsections 5.3.2, 5.3.3, and 5.3.4 analyze characteristics of a Valid and Optimal Deployment. Furthermore, Subsection 5.3.5 will determine the computational complexity of the Elementary Problem. The section concludes with an introduction of heuristic approaches that allow to approximate a Valid Deployment for the Elementary Mixed-Tenancy Problem (Section 5.3.6).

5.3.1 Definition of a Solution as a Set of Clique Covers

In order to be able to discuss characteristics of a Valid and Optimal Deployment, it is necessary to define a second representation of a Valid and Optimal Deployment based on concepts from graph theory.

In fact, for a given Elementary Mixed-Tenancy Deployment Problem, $\mathcal{L}$ shall be a solution. A solution $\mathcal{L}$ shall be defined as a set of clique covers - one per Deployment Level. Each clique cover shall be a refinement of the clique cover on lower Levels. This is expressed by the following[5].

$$\mathcal{L} = \{\mathcal{C}_{dl} \in M_{cc}(DI_{dl}) \mid dl \in DL\} \tag{5.38}$$

$$\forall dl, dl' \in DL\, \forall \mathcal{C}_{dl}, \mathcal{C}_{dl'} \in \mathcal{L} : dl < dl' \rightarrow \mathcal{C}_{dl} \sqsubseteq \mathcal{C}_{dl'} \tag{5.39}$$

Finally, the cost of a solution $\mathcal{L}$ is defined as follows.

$$w(\mathcal{L}) = \Sigma_{dl \in DL}\, w(dl) \cdot |\mathcal{C}_{dl}| \tag{5.40}$$

In order to show that the definition of $\mathcal{L}$ is, in fact, a graph-theoretic characterization of a Valid Deployment, the following Lemma is given.

Lemma 1 *Let $(T, DL, \mathcal{DI}, w)$ be any Elementary Mixed-Tenancy Deployment Problem and let $\mathcal{L}$ be any solution of it. Then, there is a Valid Deployment $(\mathcal{U}, \mathcal{F})$ for $(T, DL, \mathcal{DI}, w)$ with $w((\mathcal{U}, \mathcal{F})) = w(\mathcal{L})$.*

[5]Please note that the definitions of set M_{cc} and the refinement $\sqsubseteq$ were given in Subsection 5.1.3.

PROOF Let $\mathcal{L} = \{\mathcal{C}_{dl} \mid dl \in DL\}$ be any solution for $(T, DL, \mathcal{DI}, w)$, where $\mathcal{C}_{dl} = \{C_{dl,1}, \ldots, C_{dl,|\mathcal{C}_{dl}|}\}$ for all $dl \in DL$. The resulting Deployment $(\mathcal{U}, \mathcal{F})$ is specified as follows.

$$\forall dl \in DL : U_{dl} = \{u^1_{dl}, \ldots, u^{|\mathcal{C}_{dl}|}_{dl}\} \tag{5.41}$$

$$\forall dl \in DL \quad \forall t \in T : f_{dl}(t) = u^z_{dl} \rightarrow t \in C^z_{dl} \tag{5.42}$$

Based on this, it is easily possible to observe that $(\mathcal{U}, \mathcal{F})$ is a Valid Deployment for $(T, DL, \mathcal{DI}, w)$ and that $w((\mathcal{U}, \mathcal{F}))$ equals $w(\mathcal{L})$. ■

Summarizing the conclusions that have just been stated, it can be said that from a theoretical point of view, every Unit in a given Deployment actually corresponds to a clique in the Deployment Information graph. Thus, a solution is, in fact, a collection of clique covers, one per Deployment Level.

5.3.2 Minimal Clique Cover on High Level

As discussed in previous section, a Deployment for an Elementary Mixed-Tenancy Problem may be represented as a set of Units for every Deployment Level. Each Deployment Level's Units needs to fulfill certain constraints that were previously defined. Further, Lemma 1 showed that an alternative representation of a deployment is a set of clique covers, one per Deployment Level.

Based on this, it is possible to learn more about the structure of a Valid and Optimal Deployment. Knowing that the problem at hand is related to the clique cover problem allows the intuitive assumption that finding a minimal clique cover is somewhat beneficial towards finding a Valid and Optimal Deployment.

Constraint 4.56 defined that Deployment Information graphs of higher Levels must be subgraphs of lower Levels. This somewhat promotes the assumption that finding a minimal clique cover on the highest Deployment Level promotes the computation of a Valid and Optimal Deployment.

However, this assumption can be contradicted by investigating the example illustrated by Figure 5.7. In the figure there is an instance of an Elementary

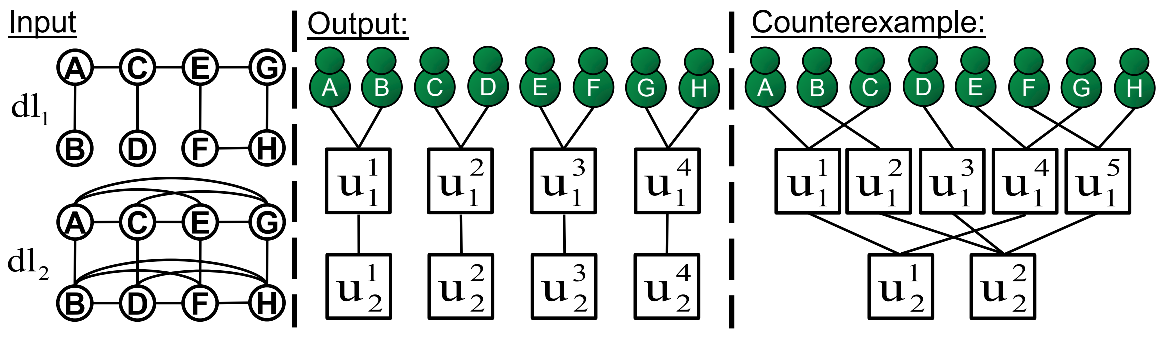

Figure 5.7.: Example not utilizing a minimal Clique Cover on highest Level

Deployment Problem illustrated. It has two Deployment Levels and a total of eight Tenants.

In the middle of the figure a Deployment is illustrated that uses a minimal clique cover on the highest Deployment Level. Based on this, a minimal number of dl_2 Units was determined that may host the created dl_1 Units. Assuming that the cost for all Units is one leads to a Deployment that causes a total cost of eight.

However, this deployment is not a Valid and Optimal Deployment since there is another Valid Deployment that uses less Units. This deployment is illustrated on the right hand side of Figure 5.7. It uses a clique cover on Level dl_1 that has one more clique than the minimal, but this leads to a reduction of two Units on Level dl_2. This way the entire Deployment only causes a total cost of seven instead of eight.

This leads to the following lemma.

Lemma 2 *There is an Elementary Mixed-Tenancy Deployment Problem* $(T, DL, \mathcal{DI}, w)$ *such that it holds: If* $\mathcal{L}$ *is an optimal solution for* $(T, DL, \mathcal{DI}, w)$*, then* $\mathcal{L}$ *cannot contain a minimal clique cover for the highest Deployment Level.*

In order to gain additional conclusions from this example, the following is a formal proof for the lemma.

PROOF Let $n \geq 2$, $X = \{x_1, \ldots, x_{2n}\}$ and $Y = \{y_1, \ldots, y_{2n}\}$. Now, the corresponding Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$ is defined as follows.

$$T = X \cup Y \tag{5.43}$$

$$DL = \{dl_1, dl_2\} \tag{5.44}$$

$$w(dl_1) = w(dl_2) = 1 \tag{5.45}$$

$$DI_{dl_1} = (V_{dl_1}, E_{dl_1}) \tag{5.46}$$

$$DI_{dl_2} = (V_{dl_2}, E_{dl_2}) \tag{5.47}$$

$$V_{dl_1} = V_{dl_2} = T \tag{5.48}$$

$$\begin{aligned} E_{dl_1} = &\{\{x_i, y_i\} \mid 1 \leq i \leq 2n\} \cup \{\{x_i, x_{i+1}\} \mid 3 \leq i < 2n\} \\ &\cup \{\{y_i, y_{i+1}\} \mid 1 \leq i < 2n\} \end{aligned} \tag{5.49}$$

$$\begin{aligned} E_{dl_2} = &\{\{x_i, y_i\} \mid 1 \leq i \leq 2n\} \cup \{\{x, x'\} \mid x, x \in X \land x \neq x'\} \\ &\cup \{\{y, y'\} \mid y, y' \in Y \land y \neq y'\} \end{aligned} \tag{5.50}$$

Figure 5.8 illustrates the graphs that have just been defined for the case that n equals 2. Furthermore, according to the definition given in Section 5.2.3 it is true that $E_{dl_1} \subseteq E_{dl_2}$.

Now, consider the following clique covers $\mathcal{C}_{dl_1}$ and $\mathcal{C}'_{dl_1}$ for the graph DI_{dl_1}.

$$\mathcal{C}_{dl_1} = \{\{x_i, y_i\} \mid 1 \leq i \leq 2n\} \tag{5.51}$$

$$\begin{aligned} \mathcal{C}'_{dl_1} = &\{\{x_1\}, \{x_2\}\} \cup \{\{x_{2i-1}, x_{2i}\} \mid 2 \leq i \leq n\} \\ &\cup \{\{y_{2i-1}, y_{2i}\} \mid 1 \leq i \leq n\} \end{aligned} \tag{5.52}$$

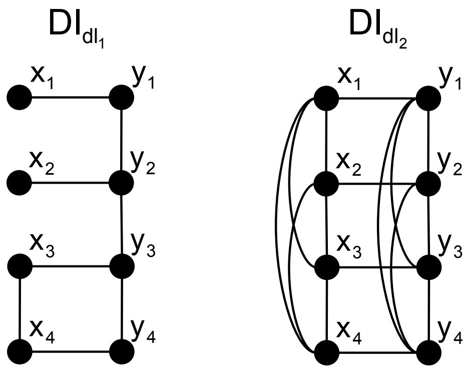

Figure 5.8.: Corresponding Graphs for $n = 2$ (Lemma 2)

For the case that n equals 2, this defines the following clique cover.

$$\mathcal{C}_{dl_1} = \{\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}, \{x_4, y_4\}\}$$
$$\mathcal{C}'_{dl_1} = \{\{x_1\}, \{x_2\}, \{x_3, x_4\}, \{y_1, y_2\}, \{y_3, y_4\}\}$$

It is quite obvious that $|\mathcal{C}_{dl_1}| = 2n$ and $|\mathcal{C}'_{dl_1}| = 2n + 1$. Moreover, it is obvious that $\mathcal{C}_{dl_1}$ is a minimal clique cover for the graph DI_{dl_1}.

Next, consider the following clique covers $\mathcal{C}_{dl_2}$ and $\mathcal{C}'_{dl_2}$ for the graph DI_{dl_2}.

$$\mathcal{C}_{dl_2} = \mathcal{C}_{dl_1} \tag{5.53}$$
$$\mathcal{C}'_{dl_2} = \{X, Y\} \tag{5.54}$$

For the case that n equals 2, this defines the following clique cover.

$$\mathcal{C}_{dl_2} = \{\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}, \{x_4, y_4\}\}$$
$$\mathcal{C}'_{dl_2} = \{\{x_1, x_2, x_3, x_4\}, \{y_1, y_2, y_3, y_4\}\}$$

Since $\mathcal{C}_{dl_1} \sqsubseteq \mathcal{C}_{dl_2}$ and $\mathcal{C}'_{dl_1} \sqsubseteq \mathcal{C}'_{dl_2}$, it is possible to immediately see that $\mathcal{L} = \{\mathcal{C}_{dl_1}, \mathcal{C}_{dl_2}\}$ and $\mathcal{L}' = \{\mathcal{C}'_{dl_1}, \mathcal{C}'_{dl_2}\}$ are solutions for the given Elementary Mixed-Tenancy Problem. Further, it can be stated that $w(\mathcal{L}) = 4n$ and $w(\mathcal{L}') = 2n + 3$.

It remains to be shown that $w(\mathcal{L}^*) = 4n$ for every solution $\mathcal{L}^* = \{\mathcal{C}^*_{dl_1}, \mathcal{C}^*_{dl_2}\}$ of the given Elementary Mixed-Tenancy Problem that meets $\mathcal{C}^*_{dl_1} = \mathcal{C}_{dl_1}$. To verify this statement, it suffices to show the following claim.

Claim. Let $\mathcal{L}^* = \{\mathcal{C}_{d_1}, \mathcal{C}^*_{dl_2}\}$ be any solution for the given Elementary Mixed-Tenancy Deployment Problem and let $C^*_{dl_2,\ell}$ be any clique in $\mathcal{C}^*_{dl_2}$. Then, it holds: $|C^*_{dl_2,\ell}| = 2$.

Since $\mathcal{C}_{dl_1} \sqsubseteq C^*_{dl_2}$, it is known that there is an i with $1 \leq i \leq 2n$ such that $\{x_i, y_i\} \subseteq C^*_{dl_2,\ell}$. First, suppose that there is any j with $1 \leq j \leq 2n$ and $j \neq i$ such that $y_j \in C^*_{dl_2,\ell}$. Since $\{x_i, y_j\} \notin E_{dl_2}$, this case cannot occur. Second, suppose that there is any j with $1 \leq j \leq 2n$ and $j \neq i$ such that $x_j \in C^*_{dl_2,\ell}$. Now, $\mathcal{C}_{dl_1} \sqsubseteq C^*_{dl_2}$ implies $y_j \in C^*_{dl_2,\ell}$. Again, because of $\{x_i, y_j\} \notin E_{dl_2}$, this case cannot occur, too. Consequently, $|C^*_{dl_2,\ell}| = 2$. ■

These results will be of importance once the Deployment Computation Algorithms of this work are going to be proposed. This will be done in Section 5.3.6.

5.3.3 Minimal Clique Cover on Low Level

In the previous section it has been shown that using a minimal clique cover on a high Deployment Level will not always result in a Valid and Optimal Deployment. However, following the idea that a minimal clique cover may actually be beneficial towards finding a Valid and Optimal Deployment, it may be assumed that using it on the lowest Deployment Level may help.

Figure 5.9 illustrates an example that proves that this assumption is also not correct for all cases. In this example there is a total of twelve Tenants that express their constraints for two Deployment Levels. On the right hand side of the figure

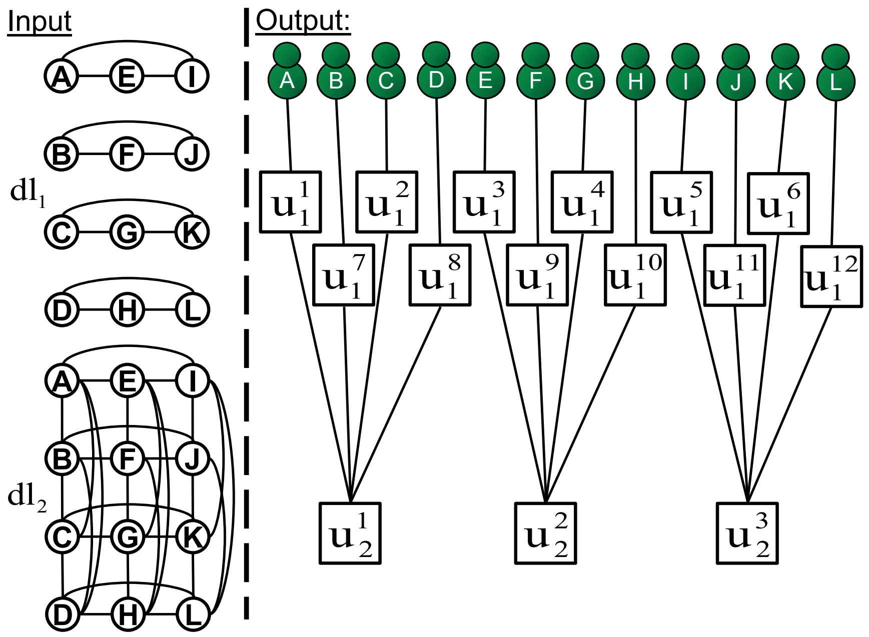

Figure 5.9.: Example utilizing minimal Clique Cover on lowest Level

a Valid Deployment is illustrated that uses a minimal clique cover on the lowest Deployment Level. This clique cover requires a total of three Units. Based on these three Units, a minimal number of possible dl_1 Units was produced. Due to the constraints of Level dl_1, this leads to a total number twelve dl_1 Units. Thus, assuming that the cost of a deployment is one, the entire Deployment causes a total cost of 15.

However, this Deployment is Valid but not Optimal. This becomes obvious if the Deployment illustrated by Figure 5.10 is considered. This Deployment uses a clique cover on Level dl_2 that contains one more clique than the minimal one. Based on this, it is possible to create four dl_1 Units to host all Tenants. Thus, the new Deployment only causes the total cost of eight instead of 15.

This leads to the following lemma.

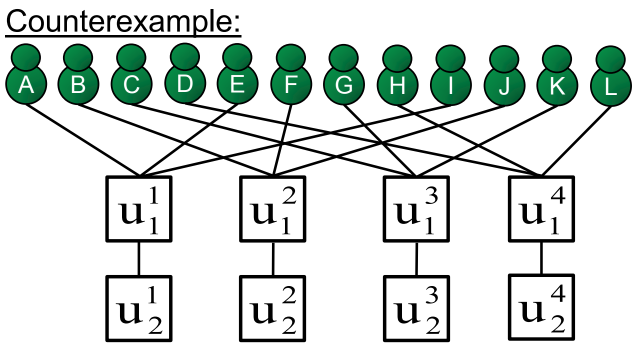

Figure 5.10.: Alternative Solution - no minimal Clique Cover on lowest Level

Lemma 3 *There is an Elementary Mixed-Tenancy Deployment Problem* $(T, DL, \mathcal{DI}, w)$ *such that it holds: If* $\mathcal{L}$ *is an optimal solution for* $(T, DL, \mathcal{DI}, w)$*, then* $\mathcal{L}$ *cannot contain a minimal clique cover for the lowest Deployment Level.*

In order to gain additional conclusions from this example, the following is a formal proof for the lemma.

PROOF Let $n \geq 4$, $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$, and $Z = \{z_1, \dots, y_n\}$. Now, the corresponding Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$ is defined as follows.

$$T = X \cup Y \cup Z \tag{5.55}$$
$$DL = \{dl_1, dl_2\} \tag{5.56}$$
$$w(dl_1) = w(dl_2) = 1 \tag{5.57}$$
$$DI_{dl_1} = (V_{dl_1}, E_{dl_1}) \tag{5.58}$$
$$DI_{dl_2} = (V_{dl_2}, E_{dl_2}) \tag{5.59}$$
$$V_{dl_1} = V_{dl_2} = T \tag{5.60}$$
$$\begin{aligned} E_{dl_1} = & \{\{x_i, y_i\} \mid 1 \leq i \leq n\} \cup \{\{x_i, z_i\} \mid 1 \leq i \leq n\} \\ & \cup \{\{y_i, z_i\} \mid 1 \leq i \leq n\} \end{aligned} \tag{5.61}$$
$$\begin{aligned} E_{dl_2} = & E_{dl_1} \cup \{\{x, x'\} \mid x, x' \in X, \, x \neq x'\} \cup \{\{y, y'\} \mid y, y' \in Y \wedge y \neq y'\} \\ & \cup \{\{z, z'\} \mid z, z' \in Z \wedge z \neq z'\} \end{aligned} \tag{5.62}$$

Figure 5.11 illustrates the graphs that have just been defined for the case that n equals 4. Consider the following clique covers $\mathcal{C}_{dl_1}$ and $\mathcal{C}'_{dl_1}$ for the graph DI_{dl_1}.

$$\mathcal{C}_{dl_1} = \{\{x\} \mid x \in X\} \cup \{\{y\} \mid y \in Y\} \cup \{\{z\} \mid z \in Z\} \tag{5.63}$$
$$\mathcal{C}'_{dl_1} = \{\{x_i, y_i, z_i\} \mid i \leq n\} \tag{5.64}$$

Obviously, $|\mathcal{C}_{dl_1}| = 3n$ and $|\mathcal{C}'_{dl_1}| = n$. For the case that n equals 4, this defines the following clique cover.

$$\begin{aligned} \mathcal{C}_{dl_1} = & \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{y_1\}, \{y_2\}, \{y_3\}, \{y_4\}, \{z_1\}, \{z_2\}, \{z_3\}, \{z_4\}\} \\ \mathcal{C}'_{dl_1} = & \{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \{x_3, y_3, z_3\}, \{x_4, y_4, z_4\}\} \end{aligned}$$

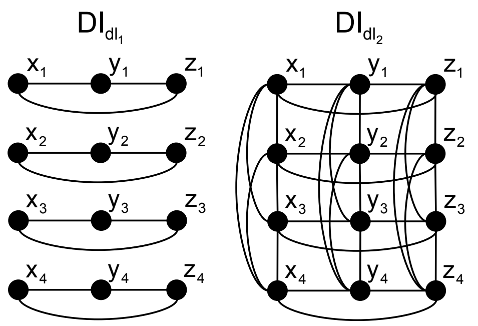

Figure 5.11.: Corresponding Graphs for $n = 4$ (Lemma 3)

Next, consider the following clique covers $\mathcal{C}_{dl_2}$ and $\mathcal{C}'_{dl_2}$ for the graph DI_{dl_2}.

$$\mathcal{C}_{dl_2} = \{X, Y, Z\} \tag{5.65}$$
$$\mathcal{C}'_{dl_2} = \mathcal{C}'_{dl_1} \tag{5.66}$$

Clearly, $|\mathcal{C}_{dl_2}| = 3$. Moreover, it is obvious that $\mathcal{C}_{dl_2}$ is a minimal clique cover for the graph DI_{dl_2}. For the case that n equals 4, this defines the following clique cover.

$$\mathcal{C}_{dl_2} = \{\{x_1, x_2, x_3, x_4\}, \{y_1, y_2, y_3, y_4\}, \{z_1, z_2, z_3, z_4\}\}$$
$$\mathcal{C}'{}_{dl_2} = \{\{x_1, y_1, z_1\}, \{x_2, y_2, z_2\}, \{x_3, y_3, z_3\}, \{x_4, y_4, z_4\}\}$$

Since $\mathcal{C}_{dl_1} \sqsubseteq \mathcal{C}_{dl_2}$ and $\mathcal{C}'_{dl_1} \sqsubseteq \mathcal{C}'_{dl_2}$, it can be concluded that $\mathcal{L} = \{\mathcal{C}_{dl_1}, \mathcal{C}_{dl_2}\}$ and $\mathcal{L}' = \{\mathcal{C}'_{dl_1}, \mathcal{C}'_{dl_2}\}$ are constitute solutions for the given Elementary Mixed-Tenancy Deployment Problem with $w(\mathcal{L}) = 3n + 3$ and $w(\mathcal{L}') = 2n$, respectively.

It remains to show that $w(\mathcal{L}^*) = 3n + 3$ for every solution $\mathcal{L}^* = \{\mathcal{C}^*_{dl_1}, \mathcal{C}^*_{dl_2}\}$ of the given Elementary Mixed-Tenancy Problem that meets $\mathcal{C}^*_{dl_2} = \mathcal{C}_{dl_2}$. To verify this statement, it suffices to show the following claim.

Claim. Let $\mathcal{L}^* = \{\mathcal{C}^*_{dl_1}, \mathcal{C}_{dl_2}\}$ be any solution for the given Elementary Mixed-Tenancy Deployment Problem and let $C^*_{dl_1,\ell}$ be any clique in $\mathcal{C}^*_{dl_1}$. Then, it holds: $|C^*_{dl_1,\ell}| = 1$.

First, consider the case that $C^*_{dl_1,\ell} \cap X \neq \varnothing$. So, let $x_i \in C^*_{dl_1,\ell}$ for some i with $1 \leq i \leq n$. By the definition of DI_{dl_1}, one immediately sees that $C^*_{dl_1,\ell} \subseteq \{x_i, y_i, z_i\}$. Now, suppose that y_i or z_i belong to $C^*_{dl_1,\ell}$. However, this implies $C^*_{dl_1,\ell} \cap Y \neq \varnothing$ and $C^*_{dl_1,\ell} \cap Z \neq \varnothing$, respectively. Thus, the constraint $\mathcal{C}^*_{dl_1} \sqsubseteq \mathcal{C}_{dl_2}$ is violated. Consequently, we may conclude that $C^*_{dl_1,\ell}$ equals the singleton set $\{x_i\}$. Note that the remaining two cases, i. e. $C^*_{dl_1,\ell} \cap Y \neq \varnothing$ and $C^*_{dl_1,\ell} \cap Z \neq \varnothing$, can be handled in a similar manner. ■

These results will be of importance once the Deployment Computation Algorithms of this work are going to be proposed. This will be done in Section 5.3.6.

5.3.4 Minimal Clique Cover on Any Level

In the previous two sections it was discussed that there are cases where a minimal clique cover on a low or on a high Deployment Level will prevent the creation of a Valid and Optimal Deployment. However, if the examples are investigated more closely, it is possible to observe that the counterexample, that proves that a minimal clique cover on the highest Level may prevent a Valid and Optimal Deployment, actually uses one on the lowest Level. The same is true for the other counterexamples that prove that a minimal clique cover on the lowest Level may prevent a Valid and Optimal Deployment. Here, a minimal clique cover is used on the highest Level.

Observing this may cause the conclusion that in any case creating a minimal clique cover for either the highest or the lowest Level will allow to find a Valid and Optimal Deployment.

However, it is quite trivial to construct a counterexample for this hypothesis as well. It is only necessary to combine the Elementary Mixed-Tenancy Problems used in the demonstration of Lemma 2 and Lemma 3. If this is done, it becomes obvious that the following lemma is true.

Lemma 4 *There is an Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$ such that it holds: If $\mathcal{L}$ is an optimal solution for $(T, DL, \mathcal{DI}, w)$, then $\mathcal{L}$ cannot contain a minimal clique cover, neither for the highest nor for the lowest Deployment Level.*

PROOF Let $n \geq 4$. Furthermore, let (T_1, DL_1, DI_1, w_1) and (T_2, DL_2, DI_2, w_2) denote the Elementary Mixed-Tenancy Deployment Problems used in the demonstration of Lemma 2 and Lemma 3, respectively, where $|T_1| = 4n$, $|T_2| = 3n$, $\mathcal{DI}_1 = \{DI_{1,dl_1}, DI_{1,dl_2}\}$ with $DI_{1,dl_1} = (V_{1,dl_1}, E_{1,dl_1})$ as well as $DI_{1,dl_2} = (V_{1,dl_2}, E_{1,dl_2})$, and $\mathcal{DI}_2 = \{DI_{2,dl_1}, DI_{2,dl_2}\}$ with $DI_{2,dl_1} = (V_{2,dl_1}, E_{2,dl_1})$ as well as $DI_{2,dl_2} = (V_{2,dl_2}, E_{2,dl_2})$.

By combining them, it is possible to create an Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$ which is defined as follows.

$$T = T_1 \cup T_2 \tag{5.67}$$
$$DL = \{dl_1, dl_2\} \tag{5.68}$$
$$w(dl_1) = w(dl_2) = 1 \tag{5.69}$$
$$DI_{dl_1} = (V_{dl_1}, E_{dl_1}) \tag{5.70}$$
$$DI_{dl_2} = (V_{dl_2}, E_{dl_2}) \tag{5.71}$$
$$V_{dl_1} = V_{dl_2} = T \tag{5.72}$$
$$E_{dl_1} = E_{1,dl_1} \cup E_{2,dl_1} \tag{5.73}$$
$$E_{dl_2} = E_{1,dl_2} \cup E_{2,dl_2} \tag{5.74}$$

The graphs that are defined by these definitions are illustrated by Figure 5.12 for the case that n equals 4. When reviewing these graphs, it is obvious that a Valid and Optimal Deployment for the entire problem will not utilize a minimal clique cover on either Levels. Thus, by applying the same argumentations as in the

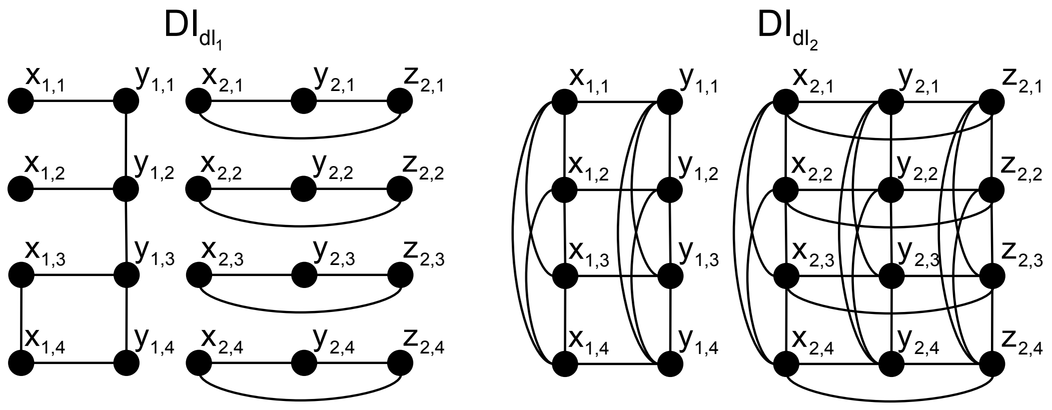

Figure 5.12.: Corresponding Graphs for $n = 4$ (Lemma 4)

proofs of Lemma 2 and Lemma 3, respectively, the correctness of the following claims can be easily verified.

Claim A. There is a solution $\mathcal{L}$ for $(T, DL, \mathcal{DI}, w)$ with $w(L) = (2n+1+n)+(2+n) = 4n + 3$.

Claim B. Let $\mathcal{L}$ be any solution for $(T, DL, \mathcal{DI}, w)$ which contains a minimal clique cover at the lowest Deployment Level. Then, it holds: $w(L) = (2n + n) + (2n + n) = 6n$.

Claim C. Let $\mathcal{L}$ be any solution for $(T, DL, \mathcal{DI}, w)$ which contains a minimal clique cover at the lowest Deployment Level. Then, it holds: $w(L) = (2n + 1 + 3n) + (2 + 3) = 5n + 6$.

■

5.3.5 Complexity of the Elementary Problem

A possible way of determining the complexity of a given problem is to reduce a problem for which the complexity is well-known to it [Wan06]. For the problem at hand, this can easily be done by considering the following argumentation.

Let $(T, DL, \mathcal{DI}, w)$ be an Elementary Mixed-Tenancy Deployment Problem with $DL = \{dl_1\}$, $\mathcal{DI} = \{DI_{dl_1}\}$ and $w(dl_1) = 1$. As already seen by Lemma 1, a solution of this problem is a clique cover in DI.

Since there is only one Deployment Level, a solution for this problem is optimal if and only if the clique cover is minimal. Thus, the problem of finding a Valid and Optimal Solution for the given Elementary Mixed-Tenancy Deployment Problem is equivalent to finding a minimal clique cover.

In subsection 5.1.3 it was discussed that finding a minimal clique cover is NP-hard, thus, it can be concluded that:

Theorem 1 *The Elementary Mixed-Tenancy Deployment Problem is NP-hard, even in the case of* $|DL| = 1$.

5.3.6 Introduction of Heuristics

In the previous subsection, the structure and complexity of a Valid and Optimal Deployment was analyzed. It was analyzed, that any Valid Deployment may be represented as a collection of clique covers, one per Deployment Level (Lemma 1). However, it was proven that there are problems where a Valid and Optimal Deployment will not utilize a minimal clique cover on the highest Level (Lemma 2), on the lowest Level (Lemma 3), or on any Deployment Level (Lemma 4).

Furthermore, it was possible to prove that the Elementary Mixed-Tenancy Deployment Problem is NP-hard and, therefore, may not be solved in a fast way. Thus, it is this section's purpose to propose two intuitive and fast algorithms that strive to approximate a Valid and Optimal Deployment. The idea of those algorithms is to utilize the known approximation algorithms that allow to approximate a minimal clique cover. Some of them were introduced in Section 5.1.3.

As stated before, it is already known at this point that they will not produce optimal results for all problems. However, their results should at least be much better than simply using the trivial solution of assigning designated Units to all Tenants. The following introduces them and analyzes their performance.

Top-Down Approach

The first heuristic that is introduced is called top-down approach. Its basic idea is to approximate a Valid and Optimal Deployment starting from the top working its way down to the lowest Deployment Level. The following is a description of the algorithm.

The first step is to construct a minimal clique cover $\mathcal{C}_{dl_1}$ in the graph DI_{dl_1}. Based on $\mathcal{C}_{dl_1}$, it is possible to define a new graph $DI_{dl_2}(\mathcal{C}_{dl_1}) = (V, E)$ by reducing the nodes in each clique of $\mathcal{C}_{dl_1}$ to one node. Furthermore, there is an edge between two of these nodes if and only if the union of the two corresponding cliques forms a clique in DI_{dl_2}. This means that for the new graph $V = \mathcal{C}_{dl_1}$ and $E = \{\{C, C'\} \mid C, C' \in \mathcal{C}_{dl_1}$ and $C \neq C'$ and $C \cup C'$ are a clique in $DI_{dl_2}\}$.

Let $\mathcal{C}'_{dl_2} = \{C'_1, \ldots, C'_n\}$ be a clique cover in $DI_{dl_2}(\mathcal{C}_{dl_1})$ and let $C_i = \bigcup_{C \in C'_i} C$ for all $1 \leq i \leq n$. Then, $\mathcal{C}_{dl_2} = \{C_1, \ldots, C_n\}$ is a clique cover in DI_{dl_2} and $\mathcal{C}_{dl_1}$ is a refinement of $\mathcal{C}_{dl_2}$.

This construction leads to the following algorithm.

Theorem 2 *For all $n \in \mathbb{N}$ there is an input $(T, DL, \mathcal{DI}, w)$ of the Elementary Mixed-Tenancy Deployment Problem such that Algorithm 1 (top-down) gives a solution with cost $4n$, whereas an optimal solution has cost $2n + 3$. Thus, the algorithm has a relative performance guarantee not better than 2.*

PROOF Consider the Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$ constructed in the proof of Lemma 2. Let DI_{dl_1} and DI_{dl_2} be the graphs with $4n$ nodes on which this deployment problem is based. In Lemma 2 it was constructed a minimal clique cover $\mathcal{C}_{dl_1}$ containing $2n$ cliques. It is easy to see that the graph $DI_{dl_2}(\mathcal{C}_{dl_1})$ forms an independent set. Hence, the top-down algorithm

Algorithm 1 Top-Down-Algorithm

```
Input: $(T, DL, \mathcal{DI}, w)$ with $|DL| = idl$
Output: $\mathcal{L} = \{\mathcal{C}_{d_1}, \ldots, \mathcal{C}_{d_m}\}$
1: Find a minimal clique cover $\mathcal{C}_{dl_1}$ of $DI_{dl_1}$
2: for $i = 2$ to $idl$ do
3:     Define the graph $DI_{dl_i}(\mathcal{C}_{dl_{i-1}})$ based on $DI_{dl_i}$ and $\mathcal{C}_{dl_{i-1}}$
4:     Find a minimal clique cover $\mathcal{C}'_{dl_i}$ in $DI_{dl_i}(\mathcal{C}_{dl_{i-1}})$
5:     Construct the clique cover $\mathcal{C}_{dl_i}$ in $DI_{dl_i}$ defined by $C'_{dl_i}$
6: end for
7: return $\mathcal{L} = \{\mathcal{C}_{dl_1}, \ldots, \mathcal{C}_{dl_{idl}}\}$
```

delivers a solution with cost $4n$. Moreover, the proof of Lemma 2 also shows that an optimal solution has cost $2n + 3$. ■

Bottom-Up Approach

The second heuristic that is presented as part of this work is called bottom-up approach. Its basic idea is to approximate a Valid Deployment starting from the lowest Deployment Level working its way up. The following is a description of the algorithm.

The first step is to construct a minimal clique cover for the lowest Deployment Level graph $DI_{dl_{idl}}$. Let $\mathcal{C}_{dl_{idl}}$ be such a clique cover in $DI_{dl_{idl}}$. The next step is to consider for each clique $C \in \mathcal{C}_{dl_{idl}}$ the graph $DI_{dl_{idl-1}}(C)$, meaning the subgraph of $DI_{dl_{idl-1}}$ induced by C.

Now, we search for clique covers $\mathcal{C}_{dl_{idl-1},C}$ in all subgraphs $DI_{dl_{idl-1}}(C)$, $C \in \mathcal{C}_{dl_{idl}}$. Let $\mathcal{C}_{dl_{idl-1}}$ be the union of all these clique covers. $\mathcal{C}_{dl_{idl-1}}$ is a clique cover of $DI_{dl_{idl-1}}$ and, by definition, a refinement of $\mathcal{C}_{dl_{idl}}$.

This construction leads to the following algorithm.

Algorithm 2 Bottom-Up Algorithm

```
Input: $(T, DL, \mathcal{DI}, w)$ with $|DL| = idl$
Output: $\mathcal{L} = \{\mathcal{C}_{dl_1}, \ldots, \mathcal{C}_{dl_{idl}}\}$
1: Find a minimal clique cover $\mathcal{C}_{dl_{idl}}$ of $DI_{dl_{idl}}$.
2: for $i = idl - 1$ to $1$ do
3:     for each $C \in \mathcal{C}_{dl_{i+1}}$ do
4:         Find a min. clique cover $\mathcal{C}_{dl_i,C}$ in $DI_{dl_i}(C)$
5:     end for
6:     $\mathcal{C}_{dl_i} := \bigcup_{C \in \mathcal{C}_{dl_{i+1}}} \mathcal{C}_{dl_i,C}$
7: end for
8: return $\mathcal{L} = \{\mathcal{C}_{dl_1}, \ldots, \mathcal{C}_{dl_{idl}}\}$
```

Theorem 3 *For all $n \in \mathbb{N}$ there is an input $(T, DL, \mathcal{DI}, w)$ of the Elementary Mixed-Tenancy Deployment Problem such that Algorithm 2 (bottom-up) gives a solution with* *cost* $3(n+1)$*, whereas an optimal solution has cost* $2n$*. Thus, the algorithm has a relative performance guarantee not better than* $3/2$*.*

PROOF Consider the Elementary Mixed-Tenancy Deployment Problem $(T, DL, \mathcal{DI}, w)$ constructed in the proof of Lemma 3. Let DI_{dl_1} and DI_{dl_2} be the graphs with $3n$ nodes on which this deployment problem is based. In Lemma 3 it was shown that there exists a minimal clique cover $\mathcal{C}_{dl_2}$ which contains 3 cliques C_1, C_2, C_3 such that $DI_{dl_1}(C_i)$ is an independent set one n nodes for every $1 \leq i \leq 3$. Hence, the bottom-up algorithm delivers a solution with cost $3n+1$. Moreover, the proof of Lemma 3 also shows that an optimal solution has cost $2n$. ■

5.4 Analysis of General Deployment Problem

In the previous section the characteristics of a Valid and Optimal Deployment were analyzed. In order to do so the Elementary Mixed-Tenancy Deployment Problem was defined and analyzed. It is a simplified version of the General Mixed-Tenancy Deployment Problem that was introduced in Section 5.2.2. Based on the conclusion gained about the structure and complexity of the Elementary Problem, this section is dedicated to analyzing the General Mixed-Tenancy Deployment Problem.

This is done by first discussing the complexity of the General Problem in Subsection 5.4.1. Based on this, the section proceeds with Subsection 5.4.2, where it will be shown that the identified characteristics of the previous section also apply to the General Problem. The section concludes with a definition of the already introduced heuristics top-down and bottom-up approaches for the General Problem. This is done in Subsection 5.4.3.

5.4.1 Complexity of the General Problem

Based on the analysis of the Elementary Mixed-Tenancy Deployment Problem, it is quite trivial to determine the complexity of the General Mixed-Tenancy Deployment Problem. The Elementary Problem was introduced as a simplification of the General Problem. The only difference between both is that the Elementary Problem is limited to only one Application Component. Further, in Subsection 5.3.5 it was proven that the Elementary Problem is NP-hard. This was accomplished by reducing the clique cover problem to it.

Thus, the complexity of the General Mixed-Tenancy Deployment Problem may be determined by the following. Let $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$ be a Mixed-Tenancy Deployment Problem with $|AC| = 1$. Then the problem is equivalent to the Elementary Deployment Problem. Thus, the following theorem can be concluded.

Theorem 4 *The Mixed-Tenancy Deployment Problem is NP-hard.*

5.4.2 Generalization of Elementary Problem

In Sections 5.3.2, 5.3.3, and 5.3.4 the characteristics of Valid and Optimal Deployment of the Elementary Mixed-Tenancy Problem were discussed. Based on the examples, it was possible to prove that there are problem instances of the Elementary Mixed-Tenancy Deployment Problem where a Valid and Optimal Deployment does not contain a minimal clique cover on the highest Level (Lemma 2), does not contain a minimal clique cover on the lowest Level (Lemma 3), and does not contain a minimal clique cover at all (Lemma 4).

The Elementary Mixed-Tenancy Deployment Problem was introduced as a special case of the Mixed-Tenancy Deployment Problem where only one Application Component exists. Because of that it is obvious that the set of all problem instances of the Elementary Mixed-Tenancy Deployment Problem is a subset of the set of all problem instances of the General Mixed-Tenancy Deployment Problem. Thus, the examples used to prove Lemma 2, 3, and 4 may also be used to prove that the same characteristics are true for the General Mixed-Tenancy Deployment Problem as well. Due to this, the following Lemmas may be stated.

Fact 1 *There is a Mixed-Tenancy Deployment Problem* $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$ *such that it holds: If* $\mathcal{L}$ *is an optimal solution for* $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$*, then* $\mathcal{L}$ *cannot contain a minimal clique cover for the highest Deployment Level.*

Fact 2 *There is a Mixed-Tenancy Deployment Problem* $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$ *such that it holds: If* $\mathcal{L}$ *is an optimal solution for* $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$*, then* $\mathcal{L}$ *cannot contain a minimal clique cover for the lowest Deployment Level.*

Fact 3 *There is a Mixed-Tenancy Deployment Problem* $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$ *such that it holds: If* $\mathcal{L}$ *is an optimal solution for* $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$*, then* $\mathcal{L}$ *cannot contain a minimal clique cover, neither for the highest nor for the lowest Deployment Level.*

5.4.3 Introduction of Heuristics

In Section 5.3.6 two heuristics were introduced that allow to approximate a Valid and Optimal Deployment for the Elementary Mixed-Tenancy Deployment Problem. This was done in a very formal way based on the formalization.

It is this section's purpose to introduce the same two algorithmic approaches for the General Mixed-Tenancy Deployment Problem. Due to the complexity of the General Problem, this time this is not done in such a formal as in Section 5.3.6. This time a description in pseudocode is provided. The advantage of this is that the algorithm is simpler, more readable and may be implemented in a standard programming language more easily.

Furthermore, instead of introducing both, the top-down approach and the bottom-up approach in separate algorithms, only one algorithm is introduced. This algorithm takes the order of Deployment Levels as input. Thus, this algorithm may approximate a Valid and Optimal Deployment following the top-down approach (using $dl_1, \ldots, dl_{idl}$ as input), following the bottom-up approach (using $dl_{idl}, \dots, dl_1$ as input), or any other required order. Even though other orders are not within the primary scope of this work, in Section 5.5.1 they will be discussed briefly within the evaluation.

In order to increase readability, the entire algorithm is split into two functions, *computeDeployment* (displayed by Algorithm 4) and *createDistG* (displayed by Algorithm 3). The *createDistG*-function is a helper function to create a so-called distribution graph based on the Deployment Information and the previously computed Units of other Deployment Levels. The basic idea of this graph is to contain higher Level Units or candidates as nodes. Further, edges are represented if two Units may be deployed together.

In the illustration of the function (Algorithm 3) lines 3 - 17 determine the relevant nodes of the distribution graph. This is done by first determining all possible candidates to become Units. It has previously been stated that for every given Deployment Level, there are certain Application Components that need to be deployed on it (earlier represented as DLS_{dl} – Subsection 4.4.8). Further, each applicable Application Component needs to be deployed for each Tenant. Thus, line 3 defines the set of all candidates as the Cartesian product of all Tenants and the Application Components applicable to the current Deployment Level.

The next step is to investigate for every candidates with which other candidate they already share a Unit on a higher Level. This is due to the requirement that once deployed together on a given Deployment Level, they must also share all Units below that for a given Application Component. The result of doing this is that the nodes of the graph will represent the candidates that are not deployed on higher Levels and all Units of higher Levels that need to be deployed on the Level currently investigated. The Units are represented as sets of candidates.

The next step towards the definition of the distribution graph is the creation of edges. There are three constraints that need to be fulfilled allowing in order to allow that two higher Level Units or candidates may be deployed on the same Unit on the current Level. Those are the following.

Application Components must be Deployable Together According to the formal problem definition given in Subsection 5.2.2, two Application Components are allowed to share a Unit on a given Level if and only if they are also deployed on the same Deployment Levels below that. In order to have readable means for expressing this, an equivalence relation was defined ($\sim_{dl}$) which is reused in line 24 of Algorithm 3 to express that.

Tenants must be willing to share Furthermore, Subsection 5.2.2 stated that Units shall only be shared by Tenants that agreed to it. This is expressed by line 25.

Units are not Deployed Separately on Lower Levels In order to have Units represent a Valid Deployment, it is necessary that Units, that are assigned to separate Units on lower Levels, are also not deployed together on higher Levels. This is expressed in lines 26 and 27.

Based on this, the *createDistG*-Function computes a distribution graph. Thus, this function is required as a helper function for the following function.

Algorithm 3 Description of createDistG-Function

```
 1: $\mathcal{V} = \{\}$
 2: **procedure** CREATEDISTG($curDL, \mathcal{DI}$)
 3:     $candidates \leftarrow \{T \times DLS_{curDL}\}$
 4:     $V_{curDL} \leftarrow \{(t, ac)\}$
 5:     **for each** $(t, ac) \in candidates$ **do**
 6:         $V_{t,ac} \leftarrow \{(t, ac)\}$
 7:         **for each** $(t', ac') \in candidates$ **do**
 8:             $V_{dl'} \leftarrow V_{dl'} \in \mathcal{V}$
 9:             **if** $\exists v : dl' < curDL \wedge v \in V_{dl'} \wedge \{(t, ac), (t', ac')\} \subseteq v$ **then**
10:                 $V_{t,ac} \leftarrow V_{t,ac} \cup \{(t', ac')\}$
11:             **end if**
12:         **end for**
13:         $V_{curDL} \leftarrow V_{curDL} \cup \{V_{t,ac}\}$
14:         $candidates \leftarrow candidates \setminus \{V_{t,ac}\}$
15:     **end for**
16:     $E_{curDL} \leftarrow \{\}$
17:     $\mathcal{V} \leftarrow \mathcal{V} \cup V_{curDL}$
18:     **for each** $v \in V_{curDL}$ **do**
19:         **for each** $(t, ac) \in v$ **do**
20:             **for each** $v' \in V_{curDL}$ **do**
21:                 **for each** $(t', ac') \in v'$ **do**
22:                     $E_{DI} \leftarrow E^{ac}_{curDL} \in DI^{ac}_{curDL} \in \mathcal{DI}$
23:                     $E_{DI'} \leftarrow E^{ac'}_{curDL} \in DI^{ac'}_{curDL} \in \mathcal{DI}$
24:                     **if** $(ac \sim_{curDL} ac') \wedge$
25:                         $((t, t') \in E_{DI} \wedge (t, t') \in E_{DI'}) \wedge$
26:                         $(\nexists v'', v''' : dl'' > curDL \wedge v'', v''' \in V_{dl''} \wedge$
27:                             $(t, ac) \in v'' \wedge (t', ac') \in v''')$ **then**
28:                         $E_{curDL} \leftarrow E_{curDL} \cup \{(v, v')\}$
29:                     **end if**
30:                 **end for**
31:             **end for**
32:         **end for**
33:     **end for**
34:     **return** $\{V_{curDL}, E_{curDL}\}$
35: **end procedure**
```

The primary function that realizes the proposed algorithm is the *computeDeployment*-function. As illustrated by Algorithm 4 it is, in fact, rather trivial and

Algorithm 4 Description of computeDeployment Function

```
1: procedure COMPUTEDEPLOYMENT($\mathcal{DI}, dlOrder, \mathcal{DLS}$)
2:     $\mathcal{U} \leftarrow \{\}$
3:     for each $cdl \in dlOrder$ do
4:         $distG \leftarrow$ CREATEDISTG($\{DI_{dl}^{ac} \mid dl = cdl \land ac \in DLS_{dl}\}, \mathcal{U}$)
5:         $\mathcal{C}_{distG} \leftarrow$ COMPUTECLIQUECOVER($distG$)
6:         $U_{cdl} \leftarrow$ CREATEUNITS($\mathcal{C}_{distG}$)
7:         $\mathcal{U} \leftarrow \{U \mid U \in \mathcal{U} \lor U = U_{cdl}\}$
8:     end for
9:     return $\mathcal{U}$
10: end procedure
```

only encapsulates one loop. Its purpose is to loop over the input that contains the order of Deployment Levels and, thus, creates the Deployment Units of each particular Deployment Level. This is done by first creating the distribution graph using the *createDistG*-function that has already been introduced. The next step is to find a minimal clique cover in this graph. In line 5 of Algorithm 4 this is indicated by a function call of the *computeCliqueCover*-function. However, this function is not discussed in any more detail since, as discussed by Subsection 5.1.3, finding a minimal clique cover is a known problem in literature where many algorithms for approximation exist – some of them were introduced in Section 5.1.3.

As discussed in Subsection 5.3.1, a Unit may in fact be seen as a clique in a graph. Thus, based on the found clique covers, the final step is to create Units by transforming the cliques into Units. This is the purpose of the *createUnits*-function. Since the function's purpose is quite obvious, there are no additional details provided about it.

Once Units were computed for all Deployment Levels, $\mathcal{U}$ will contain all Units that are required by the Deployment. Due to the way the distribution graph is created, the Units will represent a Valid Deployment for the General Mixed-Tenancy Deployment Problem according to the definition provided by Subsection 5.2.2.

5.5 Evaluation

In previous sections both the Elementary and the General Mixed-Tenancy Deployment Problem have been formalized and analyzed. Further, two heuristics, called top-down and bottom-up approach, were introduced and their performance was discussed. This allowed to gain knowledge about their worst-case performance.

However, based on these results, it is this section's purpose to do an evaluation about the presented results. It does that by first, comparing the two heuristics based on experimental data (Subsection 5.5.1). This way it shall be investigated which one performs better for realistic scenarios.

Further, the next subsection (Subsection 5.5.2) applies the better performing heuristic to the running example that was introduced in Subsection 4.6. Furthermore, the different scenarios are discussed.

The section concludes with analyzing the general efficiency of resource utilization in Mixed-Tenancy scenarios (Section 5.5.3). This is partially done based on the running example.

5.5.1 Experimental Comparison of Heuristics

In previous sections, two approaches top-down, bottom-up have been proposed to compute a Valid and Optimal Deployment. This subsection compares both approaches experimentally. The basic idea is to generate a set of samples that represent realistic problem instances. Based on these samples, Deployments are computed using both approaches. The results are compared. This is done for the Elementary and the General Problem separately. For both experiments the algorithm introduced in Subsection 5.4.3 was used.

Analysis of Elementary Mixed-Tenancy Problem

For the creation of problems of significance in the real-world, it is defined that $|T| = 50$ and $|DL| = 5$ where each Deployment Level has cost 1. Furthermore, in order to get a significant amount of problems, problems are defined where the highest graph has a density[6] in the range of 0 to 1 in steps of size 0.1. Further, it was defined that the density increases for every Level in the range of 0 to 0.5 in steps of size 0.05. This produces a total number of 121 problems. For each of these problems, 50 problem instances are randomly generated (in total $6,050$). In order to construct minimal clique covers, the greedy-algorithm Largest First for node coloring is utilized (introduced in Subsection 5.1.3).

Based on these samples, the experiments show that for about 70% of all problem instances the top-down approach outperforms the bottom-up approach. Thereby, the average improvement is about 24%. Moreover, for only about 1% of all problem instances the bottom-up approach produces better results, thereby the average improvement is about 7%. This leaves a total of 29% where both algorithms produce equal results. These results suggest to prefer the top-down approach.

However, so far the cost of Units in this experiment have been considered equal for all Deployment Levels. In real-world applications it is expected that different Deployment Levels cause different cost. Thus, it is worthwhile to investigate an approach where the costs of the Deployment Levels determine the order in which Units of the Deployment Levels are successively computed. This approach is referred to as mixed-approach.

[6] Let $G = (V, E)$ be an undirected graph with n nodes and let $x \in \mathbb{R}^+$ with $0 \leq x \leq 1$. Then G has density x, if $\frac{2|E|}{n \cdot (n-1)} = x$.

Let $(T, DL, \mathcal{DI}, w)$ be an Elementary Mixed-Tenancy Deployment Problem. The idea of the mixed-approach is to solve the problem in an arbitrary order. For example, let $DL = \{dl_1, \ldots, dl_5\}$ with $w(dl_1) > w(dl_5) > w(dl_3) > w(dl_2) > w(dl_4)$. Hence, the approach starts with determining a minimal clique cover in DI_{d_1}, then in DI_{d_5}, DI_{d_3}, DI_{d_2}, and DI_{d_4}. Obviously, such a sequence of clique covers must be a refinement to define a solution of the Deployment Problem. To ensure this, it is necessary to explain how to construct a clique cover $\mathcal{C}_{dl_j}$ in DI_{dl_j} if there are already clique covers $\mathcal{C}_{dl_i}$ in DI_{dl_i} and $\mathcal{C}_{dl_k}$ in DI_{dl_k} for $i < j < k$ and $\mathcal{C}_{dl_i} \sqsubseteq \mathcal{C}_{dl_k}$, such that $\mathcal{C}_{dl_i} \sqsubseteq \mathcal{C}_{dl_j} \sqsubseteq \mathcal{C}_{dl_k}$. For this, consider for every $C_t \in \mathcal{C}_{dl_k}$ the graph $DI_{dl_i}(C_t)$ (as in the bottom-up approach). Since $\mathcal{C}_{dl_i}$ is a refinement of $\mathcal{C}_{dl_k}$, it follows for every clique in $C \in \mathcal{C}_{dl_i}$ that either $DI_{dl_i}(C_t)$ contains C or C and $DI_{dl_i}(C_t)$ have no common node. Let $\mathcal{C}_{dl_i}(C_t)$ be a set of cliques in $\mathcal{C}_{dl_i}$ which are contained in $DI_{dl_i}(C_t)$. Obviously, $\mathcal{C}_{dl_i}(C_t)$ is a clique cover of $DI_{dl_i}(C_t)$. Hence, it is possible to construct a clique cover $\mathcal{C}_{dl_j}(C_t)$ in $DI_{dl_j}(C_t)$ as in the top-down approach. Then, $\mathcal{C}_{dl_i}(C_t)$ is a refinement of $\mathcal{C}_{dl_j}(C_t)$. Moreover, $\mathcal{C}_{dl_j}(C_t)$ is a refinement of the clique cover $\{C_t\}$, too. Now, let $\mathcal{C}_{dl_k} = \{C_1, \ldots, C_m\}$ and $\mathcal{C}_{dl_j} = \bigcup_{t=1}^{m} \mathcal{C}_{dl_j}(C_t)$. Then $\mathcal{C}_{dl_i} \sqsubseteq \mathcal{C}_{dl_j} \sqsubseteq \mathcal{C}_{dl_k}$.

In order to evaluate this idea, the same test setup as introduced above was used. In particular, the performance of the top-down, bottom-up, and all 118 remaining versions of the mixed approach were compared (118 is the result of all permutations of five Deployment Levels, $|DL|! = 5! = 120$, without top-down and bottom-up). Among these versions, the order dl_1, dl_5, dl_3, dl_2, and dl_4 is one of the best performing versions. Table 5.1 displays the results in detail.

Cases were Results are:		**Equal**	**Better (Avg.Imp.)**
Top-Down	Bottom-Up	30.7%	68.1% (23.7%)
	Best Mixed	55.5%	7.7% (4.1%)
Bottom-Up	Top-Down	30.7%	1.1% (7.4%)
	Best Mixed	32.2%	1.2% (7.5%)
Best Mixed	Top-Down	55.5%	36.8% (10.2%)
	Bottom-Up	32.2%	66.7% (27.7%)

Table 5.1.: Experimental Comparison of Approaches for Elementary Problem

Analysis of General Mixed-Tenancy Problem

For the comparison of the two heuristics on the General Mixed-Tenancy Deployment Problem $(AC, DL, T, \mathcal{DLS}, \mathcal{DI}, w_{AC}, w_{DL})$, problems are generated similarly to the way used for the Elementary Problem. In order to get problems that are realistic, it is assumed that $|T| = 50$ and $|DL| = 5$, where each Deployment Level and Application Component has cost 1. In addition, it is assumed that there are three Application Components $AC = \{ac_1, ac_2, ac_3\}$. These are deployed on the five Deployment Levels as follows: $DLS_{dl_1} = \{ac_1, ac_2, ac_3\}$, $DLS_{dl_2} = \{ac_1, ac_2\}$, $DLS_{dl_3} = \{ac_1\}$, $DLS_{dl_4} = \{ac_2, ac_3\}$, and $DLS_{dl_5} = \{ac_1, ac_2, ac_3\}$. This is the same distribution as used in the introduction example that was given in Subsection 5.2.1.

Furthermore, to generate a significant amount of problems, problems are defined where the highest graphs have a density in the range between 0 and 1 in steps of size 0.2. Further, it is assumed that density increases on every Level in the range of 0 to 0.5 in steps of 0.1. Since every combination of those densities is used for the three Application Components, this leads to a total number of $46,656$ problems ($6^3 \cdot 6^3$). Again, 50 problem instances are randomly generated for each of these problems. This leads to a total of $2,332,800$ problem instances.

Based on these samples, the experiments show that for about 67% of all problem instances the top-down approach outperforms the bottom-up approach (Elementary Problem: 70%). This is quite similar to the Elementary Problem. This time, however, there is only an average improvement of 2.4% (Elementary Problem: 24%). The bottom-up approach outperforms the top-down in 17% of all problem instances (Elementary Problem: 1%). In this case the average improvement was about 4% (Elementary Problem: 7%). This leaves a rest of about 17% of problem instances where both approaches performed equally (Elementary Problem: 29%). Even though the average improvement is not as significant as with the Elementary Problem, these results still suggest to prefer the top-down approach.

In addition to the top-down and bottom-up approach the results were also computed for the mixed-approach using the ordering $dl_1, dl_5, dl_3, dl_2, dl_4$. The approach to do that is quite similar to the one used in the Elementary Problem. Table 5.2 displays the results in detail for all three approaches. Again, the mixed-approach outperforms the other two.

Cases were Results are:		**Equal**	**Better (Avg.Imp.)**
Top-Down	Bottom-Up	16.5%	66.9% (2.4%)
	Mixed	59.7%	20.0% (3.3%)
Bottom-Up	Top-Down	16.5%	16.6% (4.1%)
	Mixed	20.4%	13.7% (3.2%)
Mixed	Top-Down	59.7%	20.3% (4.0%)
	Bottom-Up	20.4%	66.0% (2.2%)

Table 5.2.: Experimental Comparison of Approaches for General Problem

5.5.2 Application on Running Example

In Section 4.6 an example was created based on which the expressiveness of the Deployment Description Model was evaluated. The example consisted of five Application Components, five Deployment Levels, two Dimensions, thirteen Groups, and six Tenants. Furthermore, there were the following three scenarios defined according to which Tenants expressed their Constraints.

Scenario 1: All Private All Customers stated that they require their own Unit of all Application Components and all Deployment Levels.

Scenario 2: All Public All Tenants stated that they are willing to share all Application Components and all Deployment Levels with all other Tenants.

Scenario 3: Mix In this scenario each Tenant has different requirements towards the sharing of resources. These Deployment Constraints were introduced by Table 4.4.

Section 5.5.2 illustrated the Deployment Information that was created based on the example for each scenario. Based on the results of this section, it is now possible to approximate a Valid and Optimal Deployment for each of the scenarios. These are presented by Figure 5.13[7]. If the computed Deployments are compared to the corresponding Deployment Information, it is possible to observe that the Deployment realizes all Deployment Constraints and, thus, is valid. A manual proof also revealed that the computed Deployments are also optimal. For the three scenarios it is possible to gain conclusions based on each scenario separately.

Scenario 1: All Private For the first scenario resource utilization is quite the obvious. This is the scenario where the most Units are required. This is due to the fact that every Tenant requires its own set of all Units. This leads to a total number of 60 required Units.

Scenario 2: All Public As illustrated by Figure 5.13, for the second scenario, there is only one Unit required of every Application Component and Deployment Level, except for Deployment Level dl_2. There there need to be two Unit since the dl_1 Units they host rely on different stacks. Thus, the number of Tenants seems not to have an impact on the required number of Units. This leads to a total number of 10 required Units.

Scenario 3: Mix For this scenario, a total of 31 Units is required. This is less than in scenario 1 but more than in scenario 2. This is due to the fact that it was possible to have some of the Tenants share Units, however, not all of them. It must be kept in mind that assigning two Tenants to the same Unit is only possible if both Tenants agreed to it.

5.5.3 General Efficiency of Resource Utilization

In the following the efficiency of resource utilization is discussed for the General Mixed-Tenancy Deployment Problem. Mixed-Tenancy was proposed as a hybrid approach between Single and Multi-Tenancy. Thus, the minimal resource demand (lower bound) and the highest possible resource demand (upper bound) are equal to the resource demand of the Single and Multi-Tenancy deployment. These trivial bounds will be discussed in more detail by the following.

To determine the upper bound, a scenario must be considered where the maximum number of Units is required. Such a scenario dictates that every Tenant demands not to share any Unit with other Tenants (purely Single-Tenancy). It is not possible to create a scenario that requires more Units than that since every

[7]Please note that Figure 5.13 does not present the full solution for scenario 2 due to limitations of space. Thus, it only indicates a solution for one Tenant. Every Tenant has the same demand for Units in this scenario.

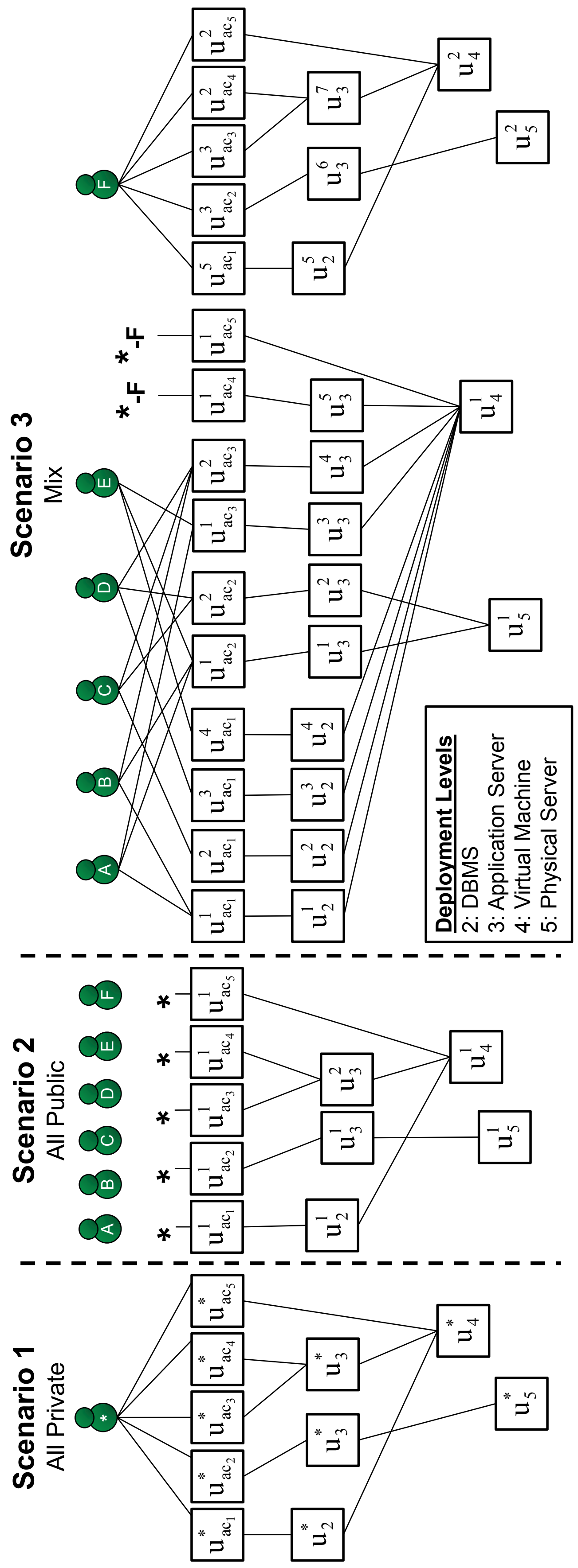

Figure 5.13.: Results of Example Introduced in Section 4.6

Tenant may only use one Unit of every Application Component and Deployment Level (other than dl_1). Furthermore, it was defined that every Unit must be used by at least one Tenant. Thus, it is possible to calculate the upper bound for the total resource demand (TRD) as follows.

$$TRD_{upperBound} = |T| \cdot \left[(\Sigma_{ac \in AC}\, w_{AC}(ac)) + (\Sigma_{dl \in DL \setminus \{dl_1\}}\, w_{DL}(dl)) \right] \tag{5.75}$$

Based on this definition, the upper bound is calculated by adding up the number of resource demands for all Application Components and the resource demand of Deployment Levels other than Level dl_1 and multiplying it with the number of Tenants. This is done since all Tenants require one Unit of each Application Component and one of every Deployment Level other than Level dl_1. An additional indication that these considerations are valid is that scenario 1 (All Private) of the evaluation example behaves accordingly.

In order to determine a lower bound for the total resource demand, a scenario may be investigated in which all Tenants are willing to share Units with all other Tenants (purely Multi-Tenancy). Since resource limitations were defined to be out of scope for this work, in such a scenario the total number of Units would be independent of the number of Tenants. It would only be necessary to create a single Unit of every Application Component and every Deployment Level other than Level dl_1. Thus, the lower bound for the total resource demand (TRD) can be calculated using the following formula.

$$TRD_{lowerBound} = (\Sigma_{ac \in AC}\, w_{AC}(ac)) + (\Sigma_{dl \in DL \setminus \{dl_1\}}\, w_{DL}(dl)) \tag{5.76}$$

Please note that for the lower bound, this is only an estimation since it may be possible that this lower bound may not be achieved for every problem. This is due to the fact that in some problems the structure of Deployment Levels and Application Components may prevent achieving this lower bound. An example for this is indicated by the example of the previous subsection. Here, the calculated lower bound would be nine Units but the minimal solution indicated by scenario 2 (All Public) requires ten Units. This is due to the fact that two application servers are required because they are deployed on a virtual machine or a physical server depending on the Application Components they host.

Besides these extremes, however, the Mixed-Tenancy approach allows different flavors in between. Obviously, every time two Tenants are agreeing to share Units, there is an opportunity for the Operator to use resource more efficiently. However, this agreement has to be mutual. In case only one Tenants agrees to share, not sharing will be allowed between both Tenants.

Thus, it can be stated that any Multi-Tenancy Deployment of an application will have at least as many Units as the lower bound but not more than the upper bound. This is also consistent with the lower and upper bound of the vertex coloring problem, as it has been discussed in Section 5.1.3.

$$TRD_{lowerBound} \quad \leq \quad TRD_{any} \quad \leq \quad TRD_{upperBound} \tag{5.77}$$

However, the conclusion gained from this is that the cost of a Deployment highly depends on the Deployment Constraints expressed by Customers. Thus, it would be in the best interest of an Operator to try to alter the Customers' behavior towards defining less restrictive Deployment Constraints. This could be achieved by creating a pricing model that offers incentives to Customers with less restrictive Deployment Constraints. However, these considerations are out of scope for this work since it focuses entirely on the technical aspects of Mixed-Tenancy.

5.6 Summary

This chapter addressed the research question of how a Valid and Optimal Deployment may be computed in a fast way (research question RQ-2). The first step towards achieving this goal was to discuss what Optimal and Valid actually means. A Valid Deployment is one that assigns Tenants to Units in a way that all Deployment Constraints provided by Customers are considered. A Valid and Optimal Deployment is a Valid Deployment that only utilizes minimal cost. In order to determine the cost of a Deployment, two types of cost were distinguished, those caused by the Tenants' resource demand and those caused by the overhead of instantiating multiple Units of the same type. It is assumed for this work that the cost caused by the Tenants' resource demand may not be altered. Thus, the actual optimization criteria is to minimize the cost that is caused by the overhead of every Unit on all Levels. Based on this discussion, a formal and complete definition of the General Mixed-Tenancy Deployment Problem was given.

Due to the complexity of the Problem at hand, a less complex version of the problem was defined that assumes that there is only one Application Component. Based on this, the so-called Elementary Mixed-Tenancy Deployment Problem, an analysis of solution characteristics was conducted. Thus, it was possible to prove that every Valid Deployment may be represented as a set of clique covers per Deployment Level. Based on this, it was obvious to assume that searching for a minimal clique cover may be beneficial to compute a Valid and Optimal Deployment. However, it was possible to contradict this assumption by proving that there is an Elementary Mixed-Tenancy Deployment Problem that cannot contain a minimal clique cover on the highest Deployment Level (Lemma 2), there is an Elementary Mixed-Tenancy Deployment Problem that cannot contain a minimal clique cover on the lowest Deployment Level (Lemma 3), and that there is an Elementary Mixed-Tenancy Deployment Problem that cannot contain a minimal clique cover at any Deployment Level (Lemma 4). Furthermore, it was possible to provide a proof that the problem is NP-hard (Theorem 1). This was achieved by investigating a version of the problem that has only one Deployment Level. In this case the problem is equivalent to the problem of finding a minimal clique cover, which is known to be NP-hard.

Knowing about the characteristics, two intuitive Deployment Computation Algorithms were proposed that allow to approximate a Valid and Optimal Deployment for the Elementary Problem. These algorithms are called top-down approach and bottom-up approach since they compute a Deployment either from the first Deployment Level down to the last or from the last up to the first. It was possible to prove that the top-down algorithm have a relative performance guarantee not better than 2 (Theorem 2) and the bottom-up algorithms has a relative performance guarantee not better than $3/2$ (Theorem 3).

Based on the analysis of the Elementary Deployment Problem, the General Mixed-Tenancy Deployment Problem was analyzed. In fact, it possible to prove that all characteristics that applied to the Elementary Deployment Problem do also apply to the General Mixed-Tenancy Deployment Problem. Further, the top-down and bottom-up approaches were introduced for the General Deployment Problem as well.

The chapter concluded with an evaluation of the presented results. First of all the two proposed Deployment Computation Algorithms, top-down and bottom-up, were compared experimentally for both problems. For the Elementary Deployment Problem the experiments show that for about 70% of all problem instances the top-down approach outperforms the bottom-up approach (average improvement is about 24%). In 29% of the problem instances both algorithms produce equal results. For the General Mixed-Tenancy Deployment Problem, the experiments show that for about 68% of all problem instances the top-down approach outperforms the bottom-up approach (average improvement of 2.4%). In 16% of all problem instances both algorithms performed equally. Thus, the experiments suggest to prefer the top-down approach.

However, for the experiments it was assumed that every Deployment Level and Application Component causes the same cost. This is an assumption that will not be true in real-world. However, the experimental investigations suggest to consider the costs of the Deployment Levels in order to determine the order in which Units of the Deployment Levels have to be successively computed. However, detailed experiments to evaluate the performance of this approach based on realistic samples is up to future research.

Furthermore, the top-down approach was applied to the running example that was introduced in the previous chapter and the general efficiency of resource utilization was discussed. Quite obviously the Mixed-Tenancy approach requires at least many resources as in a Multi-Tenancy scenario but not more than in a Single-Tenancy scenario.

The conclusion of this chapter with respect to the research questions is that research question RQ-2 cannot be solved. This is due to the fact that a Valid and Optimal Deployment cannot be found in a fast way since this problem is proven to be NP-hard. However, this chapter proposed two intuitive heuristics as Deployment Computation Algorithms. Both algorithms were analyzed from a theoretical and an experimental point of view. Based on experiments, it was possible to produce data that suggests to prefer the top-down approach.

Chapter 6
Case Study: ERP-System as Mixed-Tenancy Cloud Service

6.1. Conceptual Design of a Mixed-Tenancy Platform 135
6.1.1. Problem 1: Automated Deployment 135
6.1.2. Problem 2: Communication Mechanism 137
6.2. Selection of Cloud Application 143
6.2.1. An ERP-System as SaaS Offering - Requirements 143
6.2.2. Analysis of Available Open Source ERP-Systems 144
6.2.3. Introduction of OpenERP 145
6.3. Definition of OpenERP Scenario 146
6.3.1. Introduction of Component-based OpenERP 147
6.3.2. Selection of Application Components 149
6.3.3. Definition of Deployment Levels 150
6.3.4. Creation of Example Scenario 151
6.4. Realization of Deployment Platform for OpenERP . . . 152
6.4.1. Problem 1: Automatic Deployment 153
6.4.2. Problem 2: Communication Mechanism 155
6.5. Evaluation and Discussion of Results 156
6.5.1. Evaluation of Mixed-Tenancy OpenERP 157
6.5.2. Discussion of Resource Utilization 158
6.6. Summary 160

In theory, theory and practice are the same.
In practice, they are not.

Albert Einstein

The purpose of this chapter is to analyze the real-world applicability of the Mixed-Tenancy approach. Thus, it contributes to research question RQ-3. This is done by investigating if it is possible to deploy an existing real-world application following the Mixed-Tenancy paradigm without altering the application's functionality and behavior offered to the Customer. In order to investigate this, this chapter conducts a case study that investigates the utilization of the Mixed-Tenancy approach for the provisioning of a cloud service. In order to be able to do that, it will be necessary to investigate the challenges involved with creating a Mixed-Tenancy Deployment Platform. This contributes to research question RQ-3.1. Once the case study has been completed, it will be possible to present conclusions about the realistic resource demand of the Mixed-Tenancy approach. This contributes to research question RQ-3.2.

In this chapter the following artifacts will be created:

Mixed-Tenancy Deployment Platform A platform that allows to deploy a composite Multi-Tenancy application as Mixed-Tenancy application. It uses the Deployment Configuration as input and deploys a suitable application according to its definition.

Suitable Application A suitable application will be found and/or created that is useable in a Mixed-Tenancy environment.

The scenario that will be investigated in this chapter is that a Cloud Service Provider wishes to offer an Enterprise Resource Planning System to their Customers following the Software-as-a-Service service model.

In order to conduct the case study, this chapter is structured as follows. Section 6.1 starts this chapter by analyzing the two major problems that need to be tackled in order to be able to create a Deployment Platform. For both problems conceptual solutions are discussed. Section 6.2's goal is to identify a suitable application to be used within the case study. It does that by first defining requirements and then analyzing if existing applications can meet these requirements. Based on the identified application, Section 6.3 deals with the definition of a scenario. This includes defining the approach how Mixed-Tenancy may be introduced to the selected application, as well as the definition of Application Components, Deployment Levels, and Customer Constraints. Using the scenario, Section 6.4 describes the realization of the Mixed-Tenancy Deployment Platform suitable to deploy and run the identified application. This is done for each of the two problems that were identified. Section 6.5 will discuss the results of the case study by addressing accomplishments, shortcomings and open challenges. The chapter will be closed by a summery, that is given in Section 6.6.

Most of the results presented in this chapter were developed in cooperation with Matthias Reinhardt, Malte Rupprecht, and Björn Morr. They participated in the creation of this work in order to create their theses. Matthias wrote his master's thesis [Rei13], Malte and Björn their bachelor's theses (Malte [Rup13b], Björn [Mor14]). Matthias' thesis dealt with the creation of a Mixed-Tenancy Deployment Platform. Björn and Malte worked on the selection of OpenERP and the conduction of the case study. I supervised all three theses and guided them through the entire project.

Furthermore, the results presented in this chapter were jointly published in [Rei+14] (describing the results of Matthias' thesis) and [Rue+14] (presenting the entire case study). In addition, we received support from Brian Korduan, who is an apprentice in the area of software engineering. He helped us doing the software testing in order to evaluate the results according to my conceptual guidance.

6.1 Conceptual Design of a Mixed-Tenancy Platform

This section's purpose is to do a conceptual design of a Mixed-Tenancy Deployment Platform. There are two major jobs the Mixed-Tenancy Deployment Platform has to perform. These are explained in the following.

Automatic Deployment The platform needs to be able to create a Mixed-Tenancy deployment of a Multi-Tenancy application according to a Deployment Configuration automatically. The Deployment Configuration is created by the Deployment Configuration Generator. The Mixed-Tenancy deployment shall be provisioned automatically. This is due to the fact that this case study's purpose is to utilize the Mixed-Tenancy approach in cloud computing. For cloud computing services, high automation and minimal human interaction are major characteristics (introduced in Section 2.3.1).

Communication Mechanism As discussed earlier, this work is focused on composite applications. All Application Component instances of the same Tenant need to be able to communicate with each other. Since there may be multiple instances available of the same Application Component, the communication needs to be directed between those instances that serve the specific Tenant the request belongs to. Thus, the second job the Mixed-Tenancy Deployment Platform shall fulfill, is to manage communication between the Application Component instances.

Those two problems will be addressed by the following subsections. Each will further elaborate on the problem and discuss conceptual solutions.

6.1.1 Problem 1: Automated Deployment

This problem deals with the automated deployment of Deployment Units. Based on the Deployment Configuration, it is the Mixed-Tenancy Deployment Platform's job to realize this deployment. This requires that Units of every Deployment Level are created and provisioned automatically. In order to make that happen, it is necessary to have a steering system that is capable of executing the necessary tasks. For the remainder of this work, this steering system is called Execution Engine.

Definition 30 (Execution Engine) The *Execution Engine* is an integral part of the Deployment Platform. Its jobs are all central steering tasks.

Figure 6.1 illustrates an example in which only the two Deployment Levels, virtual machine and Application Component instances, are used. The first step to deploy a Mixed-Tenancy deployment of an application is to start deploying the

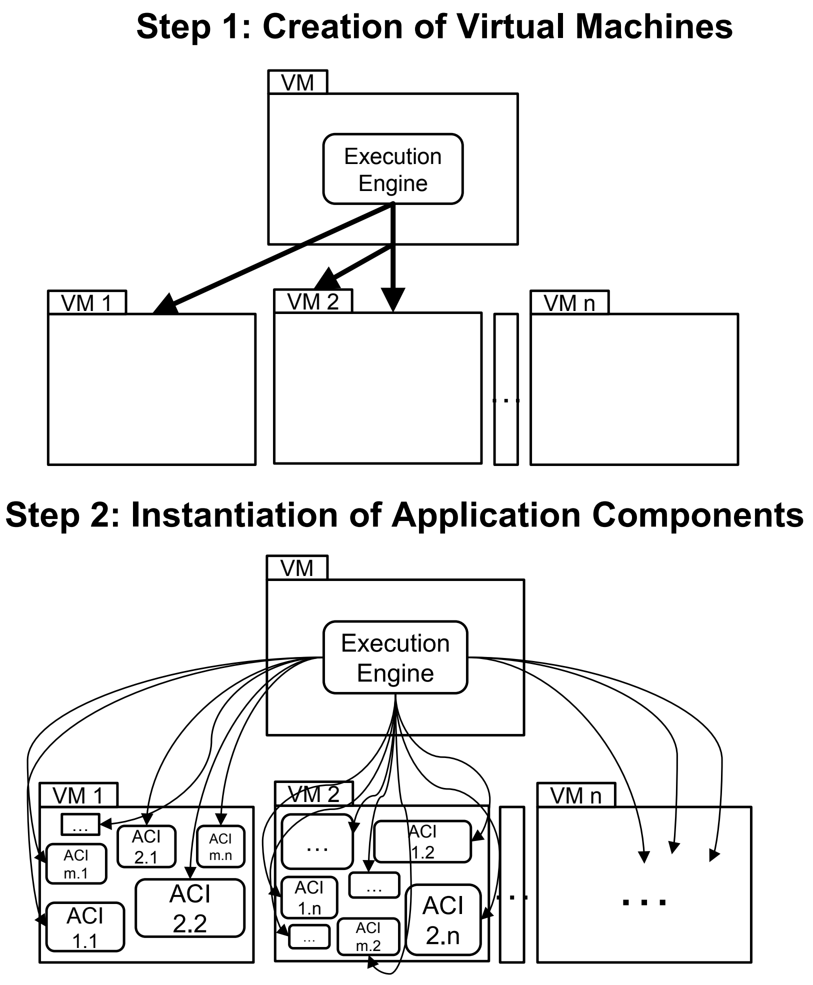

Figure 6.1.: Automatic Deployment for two Deployment Levels [Rei13]

Units of the lowest Deployment Level. In the example this is the virtual machine Level. Thus, the first step that is presented in the upper part of Figure 6.1 is to create the necessary number of virtual machines. Please note that in the figure the Execution Engine itself runs in a virtual machine. This is not a necessary requirement. The Execution Engine might just as well be run on a physical server directly. However, in order to have a homogeneous environment, within this chapter the Execution Engine is always run on a virtual machine.

Once the lowest Deployment Level has been deployed, the same has to happen to the Deployment Level(s) that run on the lowest. This way the deployment of an application is created bottom-up. This is the only way how the deployment may be created, as it was defined (in Section 4.2.1) that Units of one Deployment Level shall always run on Units of the next lower Level. For the example illustrated in Figure 6.1 there is only one additional Deployment Level - the Application Component instance Level. The second step (lower part of the figure) describes how the different Application Component instances are deployed upon the virtual machines instantiated in the previous step.

Furthermore, besides the automatic generation of Units of all Deployment Levels, the automatic deployment of the application also requires the configuration of the application. This, for example, requires the creation of all Users the Tenants require. Only then it is possible that the Tenants' Users are able to log-on to the application and use it. This setup may have to be done multiple times, once per Application Component Instance.

Furthermore, it is necessary to configure each instance the way that communication can be established. This requires that each Application Component instance knows the addresses of the other Application Component instances to be able to communicate. However, with this point, there are more challenges involved that will be discussed in the next subsection.

6.1.2 Problem 2: Communication Mechanism

It is the goal of this section to create a communication mechanism that allows the platform to establish communication between Application Component instances of an existing Multi-Tenant composite application. What this means is illustrated by Figure 6.2. In this figure, three Tenants (Tenant A = red, Tenant B = green,

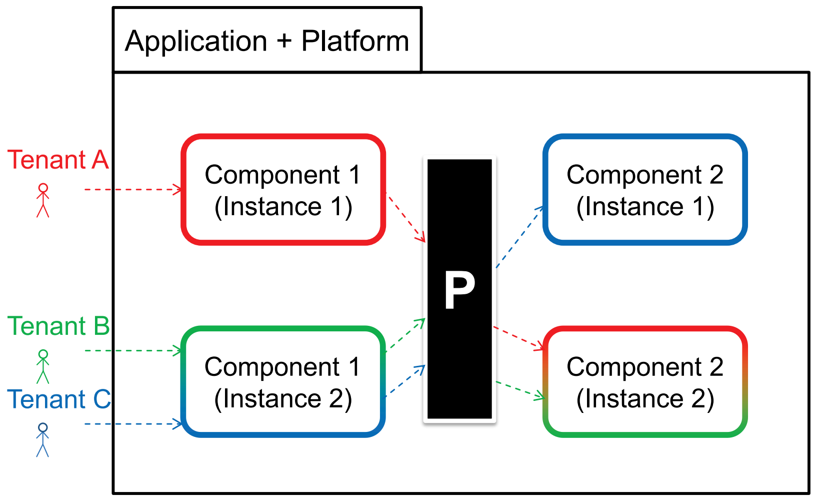

Figure 6.2.: Overview of Communication Problem [Rei13]

Tenant C = blue; P = Platform) are using an application that is composed of two Application Components. There are two instances deployed of each Application Component. The instances' border color indicates which Tenants are allowed to use which instance (e.g. AC-1 Instance 2 is used by green = Tenant B and blue = Tenant C).

Once Application Component 1 requires communication with an instance of Application Component 2, the communication needs to be directed to the correct instance of Application Component 2. This direction of communication needs to be established based on the Tenant triggering it. For Application Component 1 Instance 2, for example, a communication to Application Component 2, may have to go to Instance 1 or to Instance 2 depending on the Tenants.

From this discussion the following two requirements are conducted that need to be fulfilled by a suitable communication mechanism.

Decentralized Communication The goal of the novel approach of Mixed-Tenancy was to allow Customers to express Deployment Constraints that restrict with whom they share resources. Based on these Deployment Constraints the computed Valid Deployment was provisioned – as discussed by the the previous subsection.

However, the Deployment Constraints shall also be enforced for the communication between Application Component instances. This means that Tenants that do not wish to share resources shall also not be able share the same platform components (e.g. the Execution Engine) that establish communication. This may be realized by creating a communication mechanism that is distributed without having a central system used by all Tenants. This idea is further elaborated when discussing the individual candidate patterns.

Separation of Platform Logic It is the aim of this chapter and research question RQ-3 to evaluate if the Mixed-Tenancy approach is applicable to existing applications. Thus, the Mixed-Tenancy Deployment Platform must have the ability to deploy existing applications according to the Mixed-Tenancy paradigm. In order to have minimal effort needed to migrate an existing application onto the Mixed-Tenancy Deployment Platform, it would be good if the existing application's source code would not have to be altered. This goal can by achieved by separating the application logic from the platform logic. This is further elaborated in the following.

In the following different patterns will be introduced and their shortcomings will be discussed. Figure 6.3 gives an overview of these patterns by illustrating their structure as well as their sequence (UML sequence diagrams). In order to do that, each pattern is applied to the same example that shall illustrate the behavior of the pattern. In this example there are two Tenants and three Application Components. Of Application Component 1 and 2 there are designated instances for both Tenants. However, there is only one instance of Application Component 2 that both Tenants share. Furthermore, each pattern relies on the Execution Engine. The Execution Engine is the central entity that holds the Deployment Configuration. Thus, it knows which Tenants are allowed to communicate with which Application Component Instances. However, the Execution Engine will also take over other jobs depending on the pattern.

Delegator

The first and very obvious pattern identified was called delegator. The structure and behavior of the pattern is illustrated by 6.3 (left). Within this pattern the Execution Engine's job is to delegate communication.

When a User or an Application Component instance wants to call a function of another Application Component, it asks the delegator which instance of that

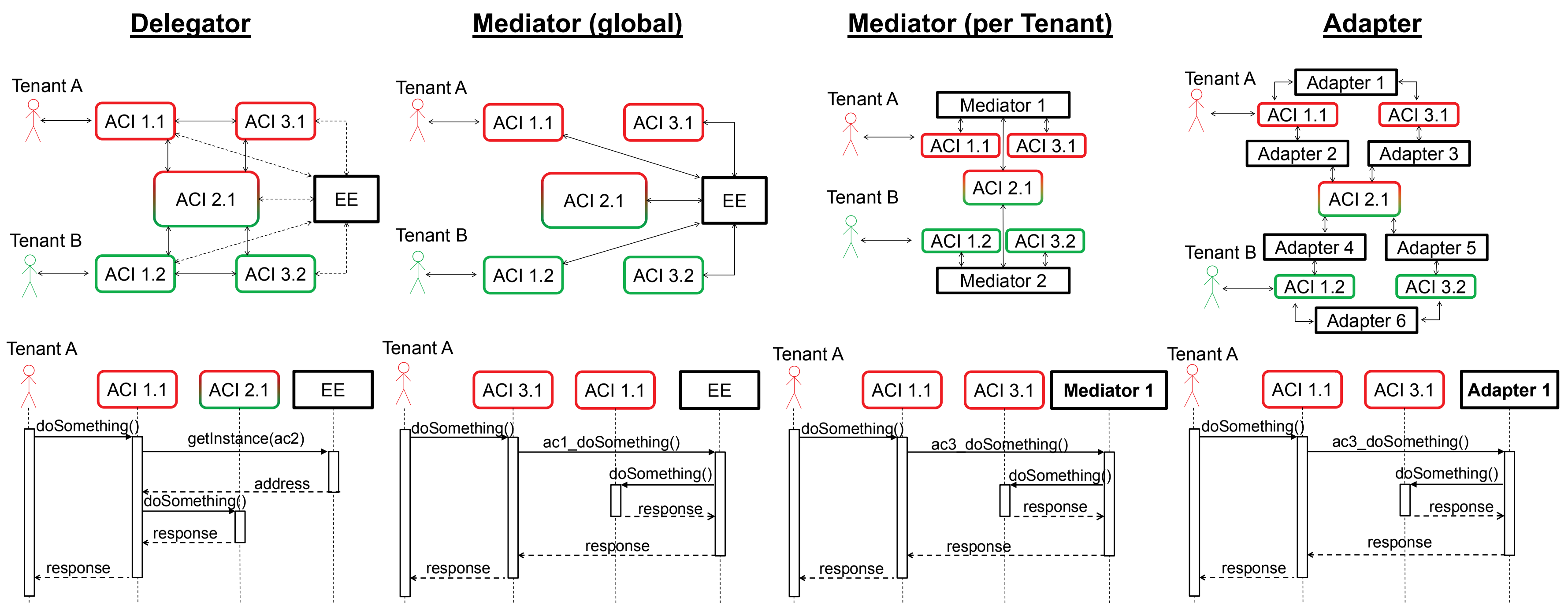

Figure 6.3.: Overview of Pattern to Tackle the Communication Problem [Rei+14].

Application Component belongs to the current Tenant. This control communication is represented by the dashed arrows in Figure 6.3. The Execution Engine responds with the whereabouts in form of an address. Thereby, a contact can be established between the two instances or the User and the instance. This direct communication is represented by solid lines. Change of the location is not an issue, because the contact to the Execution Engine will happen before every function call of another Application Component instance.

With respect to the requirements, there is a problem with the separation of the platform logic. The single instance of Application Component 2 needs to know multiple instances of another Application Component, because it is shared by multiple Tenants. Thus, platform logic is needed in the Application Component, which is not desired because it complicates the adaptation process of an application to the platform. However, the pattern meets the requirement for decentralized communication and, thereby, applies to Customers' Deployment Constraints.

Mediator – Global

Secondly, the mediator pattern provided by [GHJ94] may be utilized. The Mediator is an object which encapsulates the interaction between an amount of objects. Thus, the objects are able to communicate without directly connecting to each other. Based on this investigation, two ways to implement the pattern emerged – global mediator and one mediator per Tenant.

The global mediator is characterized by a single mediator instance which is represented by the Execution Engine. The structure and behavior of the pattern is illustrated by 6.3 (second left). In this figure, the solid lines between the Application Component instances and the mediator indicate that data communication between the Application Component instances passes through the global mediator This is unlike the delegator pattern, where the data communication happens directly between the Component instances.

With the use of the global mediator, there is no communication logic inside the Application Components. They just have to be able to communicate with the Execution Engine. Thus, there is no problem with the special case of Application Component 2, where one instance is used by two Tenants at once. But because of the passing of the complete data communication through the Execution Engine, there is a problem with the other requirement. If the Customer requested the separation from another Customer, it is not legitimate to redirect both their data communication through a common point of intersection, which here unfortunately is the case. Another negative aspect is a possible overload of the global mediator in a larger scenario, which could slow down the system as the global mediator becomes a bottleneck for all communication.

Mediator – per Tenant

As already mentioned, there is a second way to implement the mediator pattern. The structure and behavior of this is illustrated by Figure 6.3 (second right).

The idea is to create multiple mediator objects, one mediator per Tenant. In contrast to the global mediator the data communication which passes through a mediator is related to the same Tenant. Because of this, the requirement of decentralized communication has successfully been respected.

In case of Application Component 2, however, there is only one instance used by both Tenants. The instance has to decide which mediator to contact if it wants to call a function of another Application Component. Clearly, there is communication logic necessary inside the Application Components, which stands in conflict with the second requirement.

Adapter

Another classic pattern given by [GHJ94] is the adapter pattern. Usually it will be applied to establish communication despite incompatible interfaces. For example, when it is required in a project to include third-party components, without the possibility to adjust the interfaces, an adapter is an appropriate solution.

In Figure 6.3 it is illustrated that there exists an adapter between every two Application Component instances that will eventually need to communicate. Such adapters encapsulate the functionality of their related Application Component instances.

This setup follows the requirement of decentralized communication. All data communication passing through an adapter is related to the same Tenant. With respect to the second requirement a violation may be noticed for Application Component 2. A mutually used Application Component instance must decide which adapter to address. This decision has to be made depending on the Tenant currently accessing the Application Component. Thus, communication logic is necessary inside the Application Components.

Summary and Introduction of Connector Pattern

In the previous paragraphs, multiple patterns were introduced and analyzed in order to find out if they would satisfy the requirements for a communication mechanism. A summary of the investigated patterns (Delegator, Mediator - Global, Mediator - per Tenant, and Adapter) and their applicability is shown in Table 6.1. None of the aforementioned patterns matches both previously defined

	Patterns				
Requirements	Delegator	Mediator - Global	Mediator - per Tenant	Adapter	Connector
Decentralized Communication	✓	✗	✓	✓	✓
Separation of Platform Logic	✗	✓	✗	✗	✓

Table 6.1.: Summary of presented Patterns

requirements. Thus, a new pattern, that combines the positive aspects of the previous patterns, needs to be created. Due to its behavior, it is called Mixed-Tenancy connector pattern.

The left side of Figure 6.4 describes the structure of the pattern using the same example as for the other patterns. For every Application Component instance there is a corresponding connector element. The Application Component instances are solely able to initiate a connection to their connector. This is geared to the global mediator pattern, where also no platform logic was located in the Application Component instances. In contrast to the global mediator, this approach is distributing the platform intelligence into multiple connector elements and not into a single object like a mediator. Besides, there is a direct communication between the connector elements. This works just as in the delegator pattern. On the right hand side of Figure 6.4 it is demonstrated how the com-

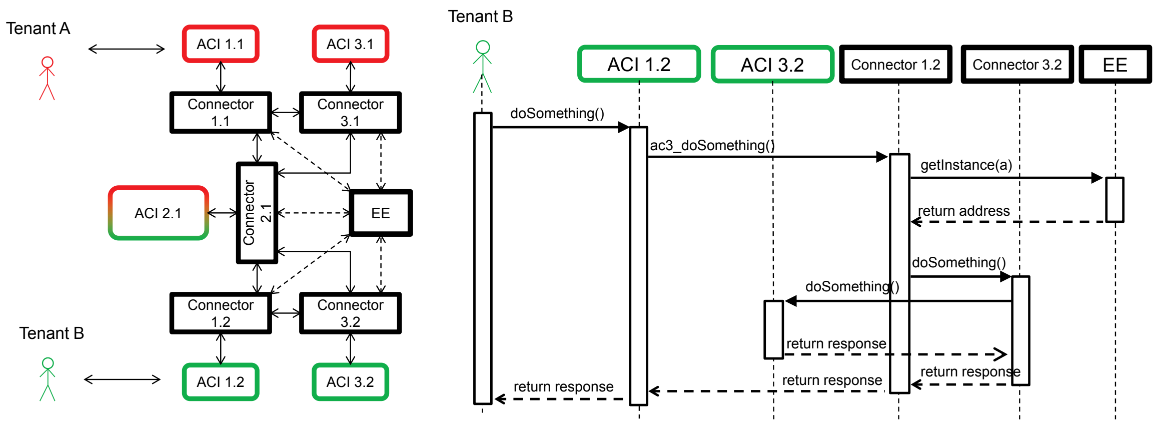

Figure 6.4.: Overview of the Connector Pattern [Rei13]

munication works. If an Application Component instance, respectively a User, wants to retrieve data from another Application Component, it is calling the specific function at the related connector. Thus, here is no platform logic necessary. The Application Component instance treats the function just like a local one. The connector is now handling the establishment of the connection. The Execution Engine acts just like in the delegator pattern. The only difference is, that the connector is requesting the address or the reference of the corresponding connector of the target Component instance – not of the Component instance itself. After the actual function call took place, the result will be returned to the Application Component that was initially calling. As illustrated by the figure, a connector encapsulates the needed functions of all other components for the related Component instance and the provided functions to be called by the other Component instances. This is included the enforcement of the Deployment Constraints on a communication level

The connector pattern satisfies both previously defined requirements. This is also indicated in Table 6.1. It will be used to create a Mixed-Tenancy Deployment Platform that allows both, to deploy existing Multi-Tenancy composite applications and to enforces Deployment Constraints also on communication level.

6.2 Selection of Cloud Application

This section's purpose is to select an example application that is going to be used for this case study. As the primary focus of this chapter is to evaluate the applicability of the Mixed-Tenancy approach to real-world applications, this section is very important as it will analyze to which applications the Mixed-Tenancy approach is applicable and if they may be found in real-world.

In order to do that the section starts by discussing the domain from which an application shall be selected and expresses requirements it shall fulfill (Section 6.2.1). Based on this, the available applications from the chosen domain shall be analyzed and selected (Section 6.2.2). The section concludes with a discussion of the selected application (Section 6.2.3).

6.2.1 An ERP-System as SaaS Offering - Requirements

As already stated in the introduction of this chapter, the scenario that is about to be investigated in this chapter, is that a Cloud Service Provider wishes to offer an Enterprise Resource Planning (ERP) System as a Software-as-a-Service offering to their Customers. The reason for this case study to focus on ERP-Systems is that Mixed-Tenancy would allow Customers to specify their Deployment Constraints for the individual modules or sub-systems of the ERP-System (e.g. Controlling, Accounting, Procurement, etc.). Furthermore, there may be some Customers for which an ERP-System, provided to them following the cloud paradigm, may be very attractive since they will not have to provide their own IT-infrastructure but get the entire application as a service from the cloud service provider. This may especially be attractive for small and medium sized enterprises that do not have their main area of business in the IT industry.

In order to be able to conduct the case study for this scenario, it is necessary to select an application. The first step towards this is to define requirements that shall be met by a suitable application. Since this work has a technical focus, the definition of requirements does not cover any commercial interests, but is limited to technical aspects. The following are mandatory requirements for selecting an application that may be deployed according to the Mixed-Tenancy paradigm.

Web-based If an application has a web-based user interface, Customers may use the application without installing any special software on their client machines. All they need is a standard web-browser that is pre-installed on almost all desktop workstations today. Furthermore, a web-based user interface gives additional freedom to Users as to from where and with which device they use the application.

Multi-Tenancy The Mixed-Tenancy requires that an Application Component is capable of handling multiple Customers in a Multi-Tenancy fashion. Thus, it is a requirement that the application, that is selected within this case study, has this ability.

Component-based One of the major advantages of the Mixed-Tenancy approach is that Customers may express their Deployment Constraints not just for the entire application but for each Application Component. This is why it is an essential requirement that an application that shall be provisioned following the Mixed-Tenancy approach is component-based. What this work defines as component-based has been introduced in Subsection 2.1.2. Again, this definition includes that individual Application Components, that compose the application, may be deployed independently from each other.

Open Source As stated in Section 6.1.2, it is the goal of this work to provision a application without having to alter the application's code base. However, when the conduction of the case study was started it was not clear if this goal may actually be achieved. Thus, the selection of applications shall be limited to those where the code base is accessible. An additional reason to consider only applications available under open source licenses is that those applications do not require an investment into licenses to use the application.

6.2.2 Analysis of Available Open Source ERP-Systems

Based on the requirements that were introduced in the previous section, it is the goal of this section to select the most suitable application from the pool of available ERP-systems. The first thing to do is to identify the available ERP-systems currently available and being used in the real-world.

Since one of the requirements is that the ERP-system is open source, the research may be focused on those. Two intensive reports on available open source ERP-systems were found. [hei09] dates back to the year 2009 and enlisted a total of six such systems. The second report was published by the Fraunhofer Society in 2011 [SES11] and lists a total of fourteen systems including the six of [hei09]. These systems are ADempiere [ADe14], Apache Ofbiz [Apa14], AvERP [SYN14], CAO-Faktura [CAO14], Compiere [Com14], Limbas [Lim14], Lx-Office [Kiv14], Openbravo [Ope14a], OpenERP [Ope14], Opentaps [Ope14b], SQL-Ledger [SQL14], Tryton [Try14], WebERP [web14], and xTuple [xTu14]. In the time when the search was conducted, no other reports were found reporting more ERP-Systems.

In a first phase these fourteen systems were superficially evaluated about how well they may satisfy the given requirements. This superficial evaluation was conducted based on the information available on their product/project websites, communities, online fora, mailing lists, and the two reports themselves. Based on this first evaluation, it was possible to limit the number of potentially suitable applications down to six. These are ADempiere, Tryton, Openbravo, OpenERP, Apache OFBiz, and Opentaps. These six systems were evaluated further in a second round of evaluation.

This time, each of these systems was installed in their regular (non-Mixed-Tenancy) way. Based on the experience gained during installation, a more thorough evaluation of the application was performed. Thus, it was investigated if these systems may be useable for the scope of this case study. The result is illustrated by Table 6.2. Based on the table, it is possible to say that only OpenERP,

	Requirements			
System	Web-based	Multi-Tenancy	Component-based	Open Source
ADempiere	✓	✓	✗	✓
Tryton	✗	✓	✓	✓
Openbravo	✓	✓	✗	✓
OpenERP	✓	✓	✓	✓
Apache OFBiz	✓	✓	✓	✓
Opentaps	✓	✓	✓	✓

Table 6.2.: Details about Relevant Applications

Apache OfBiz, and Opentaps may be relevant candidates for this case study.

These three systems were investigated even further. This time they were investigated on a code base level in order to determine if they may be useable. It must be noted at this point that Apache OFBiz and Opentaps actually share a significant part of their code base. When doing the code analysis for these two systems, it was apparent, that even so they claim to be component-based, they are in fact not according to the definition of this work. It is true that they arrange their classes in so called *component sets* that manifest themselves in a folder in the file system and a component definition in an XML-file. However, between the classes of different *component sets* there is very high coupling through Java-import commands. Based on the analysis conducted by [Rup13b], a Mixed-Tenancy deployment of neither of those applications is possible due to this high coupling.

OpenERP, on the other hand, still seemed promising after a code review since at this point it seemed to be fulfilling all previously defined requirements. This is why for this work OpenERP has been selected for the usage within the case study.

6.2.3 Introduction of OpenERP

OpenERP is, as the name already suggests, an open source enterprise resource planning system. As for many open source applications, behind OpenERP stands a company. In case of OpenERP it is OpenERP a.s., a company from Belgium that was founded in 2005. Nevertheless, OpenERP's entire source code is available to the public under the AGPL license. OpenERP a.s. offers additional services like maintenance and support [Ope14].

OpenERP has won the annual open source award BOSSIES (Best of Open Source Software) of InfoWold.com twice in a row in 2012 [Inf12] and 2013 [Inf13].

OpenERP itself is built based on a highly modular architecture. This means that OpenERP, in fact, only becomes a useable application that offers functionality if modules are loaded. This allows Customers to have OpenERP cover only the functionality they actually require. Currently, there are more than 200 such modules available for usage in the standard bundle that is available for download. OpenERP implements Multi-Tenancy following the *separated database* method that was introduced in Section 3.1.1. It is primarily written in the programming language Python.

OpenERP itself, without any modules, only provides facilities for the most basic things that are used by modules. For example, it provides facilities for interacting with a database in form of a full object-relational mapper. The database used by OpenERP is PostgreSQL. In addition, it provides communication mechanisms to establish communication between modules, and the OpenERP system provides the web interface[1]. The web interface is a rich client application that is completely developed in JavaScript. It interchanges data with the server through XML-RPC and JSON-RPC services.

Modules are implemented to be stateless. They gain access to the provided facilities by inheritance of a class called *Base_Model*. If, for example, module A needs to communicate with module B by inheriting from *Base_Model*, module A gains the functionality to request an instance of module B from the central module registry, simply by calling a function. The OpenERP system maintains the registry with instances of all installed modules.

Since the architecture of OpenERP dictates that each module is run in the OpenERP system, it is not possible to deploy individual modules independently from each other. This is in violation with the definition of an Application Component used by this work. However, since OpenERP seems to be the best possible application available, the following section will introduce the scenario how a case study may still be conducted.

6.3 Definition of OpenERP Scenario

In the previous section, OpenERP was selected as an example application for the case study of this work. Based on this, it is this section's purpose to introduce a scenario in which Tenants are using OpenERP in a Mixed-Tenancy setup.

In order to do that, the section is structured as follows. It starts by analyzing how the Mixed-Tenancy approach may be introduced to OpenERP (Subsection 6.3.1). Based on this, Subsection 6.3.2 will discuss which Application Components were used for the scenario. Once this is done it is possible to introduce the Deployment Levels that shall be covered in the investigated scenario (Subsection 6.3.3). Finally, Subsection 6.3.4 discusses the definition of Tenants and their Deployment Constraints.

[1]In addition to the web interface, OpenERP also offers an additional non-web client. However, since version 7 of OpenERP it has been recommended to use the web-client. Thus, the old one is not considered relevant for this work.

6.3.1 Introduction of Component-based OpenERP

As discussed in the previous section, it is not possible to apply the Mixed-Tenancy approach directly to OpenERP since the OpenERP's architecture does not consist of Application Components according to this work's definition. It is this subsection's purpose to discuss how it is still possible to apply the Mixed-Tenancy approach to OpenERP. In order to do that, a minimal example is discussed.

In this minimal example, there are four modules that shall be deployed separately from each other. These are the modules idea, email_template, mail, and base_setup. In Figure 6.5 a standard (non Mixed-Tenancy) deployment is illustrated. In this figure, it is illustrated that the previously mentioned four modules

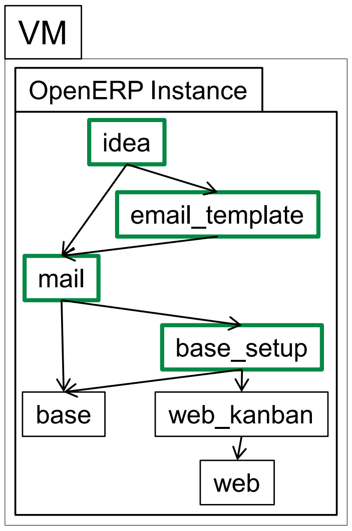

Figure 6.5.: Example Installation of OpenERP with 4 Modules [Rup13b]

are deployed in a single OpenERP system. This OpenERP system runs in a virtual machine. Besides the previously mentioned four modules, three additional ones are added. These three are mandatory modules since they deal with things that are required by all other modules (e.g. user management, module catalog management, user interface).

Communication between these seven modules is established using the mechanism that was introduced in the previous section. Each of those modules extends a class called Base_Model. Thus, it has the functionality to get instances of another module, on which it may directly call functions. This way, communication between modules can be established.

Figure 6.6 illustrates how the four modules may be deployed separately. As visible in the figure a virtual machine hosts four instances of the OpenERP system this time, instead of just one. Each of these instances hosts one of the four modules (marked green) and the three mandatory modules. Further, the other three modules are replaced by a connector according to the pattern that was introduced in Section 6.1.2. This means that an Application Component, in the sense it has been used in this work so far, is in fact an instance of OpenERP with only one module, multiple connectors and some mandatory modules.

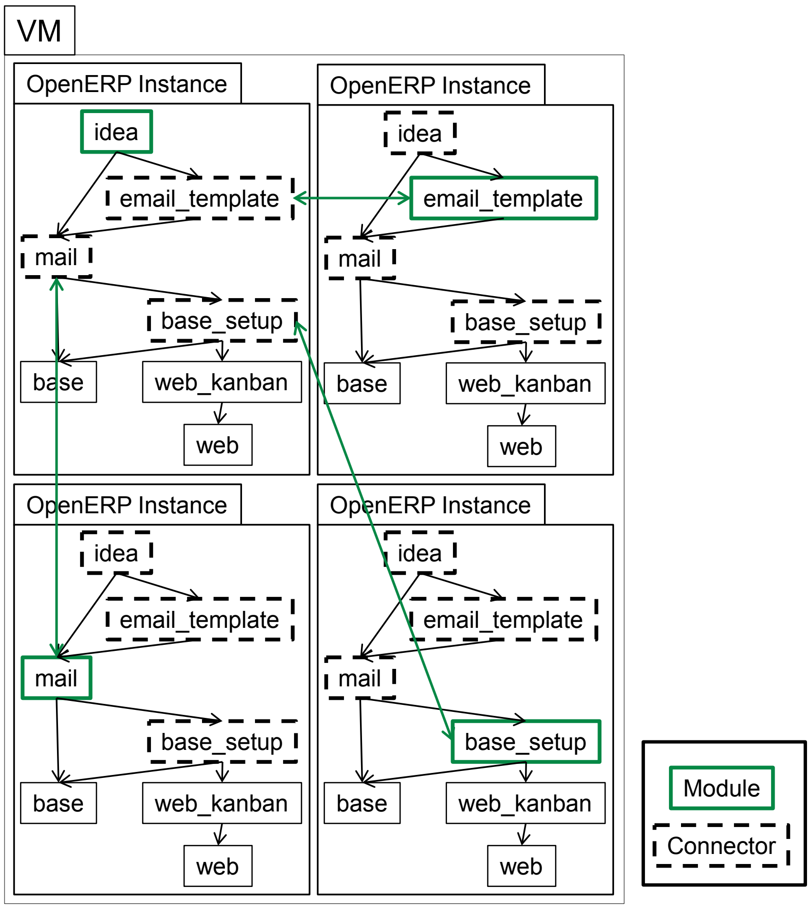

Figure 6.6.: Mixed-Tenancy Installation of OpenERP with 4 Modules [Rup13b]

Even in the Mixed-Tenancy deployment, modules need to communicate among each other. This is done following the connector pattern. If a module needs to communicate with another, it uses the mechanism offered by Base_Module to gain an instance of the other. However, this time it does not get an actual instance of the module to be called but only of its connector. The connector has the same signature as the module that is supposed to be called. Thus, the calling module does not need to be changed. Once the calling module calls a function, the connector will determine where the actual instance of the module is running and will call it using the XML-RPC service each instance OpenERP already offers. The answers of the actual module are then passed back to the calling module.

Splitting OpenERP in the way that has just been described allows to deploy modules individually. This makes the altered version in fact a component-based version of OpenERP, thus, it is from now on referred to as *component-based OpenERP*. However, it produces an overhead since it is necessary to deploy the mandatory modules redundantly and include the connectors.

6.3.2 Selection of Application Components

The following describes how the Application Components used in the case study were created.

The first Application Component that is supposed to be defined is the database. This allows Customers to express their Deployment Constraints for using the database.

Furthermore, it is the idea of this work that some of the modules of OpenERP shall become Application Components. It would be possible to create an Application Component for every module that shall be part of OpenERP. The approach making that possible was introduced by the previous section. However, the standard version of OpenERP, that is available from the project's website, contains a total of about 200 modules. This would not be practical due to two reasons. First of all it would cause significant description effort for the Customer and, secondly, it would cause very high redundancy due to the way Application Components are created for the component-based version of OpenERP. Furthermore, many of those 200 modules are, in fact, very technical and it would not make much sense for Customers to have the ability to express their Deployment Constraints for those.

Due to this, for the case study of this work, multiple OpenERP modules were bundled to an Application Component. The first thing that is important to understand is that there are different kinds of modules within OpenERP – regular modules and apps. From a technical perspective the only difference between an app and a module is a flag in its description. However, this flag's result is that apps are very visible to the Tenants since they are selected and installed by Tenants. Example apps that come with the standard version of OpenERP are customer relationship management or human resources. They encapsulate bigger portions of functionality. In fact, they often require multiple regular modules. Thus, one Application Component was created for every OpenERP app. In addition, each Application Component contains those modules that are required by the OpenERP app. This results in a total of 23 Application Components. When analyzing the dependency structure between apps and modules within OpenERP, it became apparent that there may be multiple apps that require the same module. It was decided that in these cases separate copies of the same module would be included in both Application Components. This ensures that data that is processed by one app is only processed by modules that are deployed according to the same Deployment Constraints.

However, when the Application Components were analyzed, it was apparent that there were major redundancies between them. It was possible to reduce them significantly by creating three additional Application Components called account, portal and procurement. A detailed description of all Application Components and the modules they contain can be found in Appendix A.1. Once all 27 Application Components (26 from modules and 1 from the Database) were created, about 65 modules were assigned to them. From the original 200 modules that left 135 not assigned to Application Components. These modules cover optional functionality. A major part of them, for example, covers localization issues like language of the user interface or introducing country specific aspects to the accounting app. Further, they cover optional functionality that allows to integrate OpenERP with other systems like Microsoft Outlook or Mozilla Thunderbird. Since the goal of this case study is to provide a proof of concept, the optional modules were not included in the case study. Further, from a technical perspective the 65 modules that are covered contain more than 65% of the source code from the 200 modules[2]. Thus, it is assumed that the case study contains the major part of the functionality of OpenERP.

Based on the discussion that has just been presented, it can be concluded that for component-based OpenERP a total of 27 Application Components shall be considered within this case study. 26 of them are related to apps or modules of OpenERP, the 27th is the database. These 27 Application Components cover all apps that belong to the standard version of OpenERP and the modules they require.

6.3.3 Definition of Deployment Levels

The next step towards the definition of an example scenario is the definition of the Deployment Levels that shall be considered. It has previously been stated that in the scenario that there shall be a total of 27 Application Components defined. In fact 26, of them are Application Components that realize OpenERP apps and one which is the database.

In the scenario to be investigated, these two types of Application Components shall require different stacks. However, both, modules and the database shall be deployed on a virtual machine. In order to cover all this, the following Deploy-

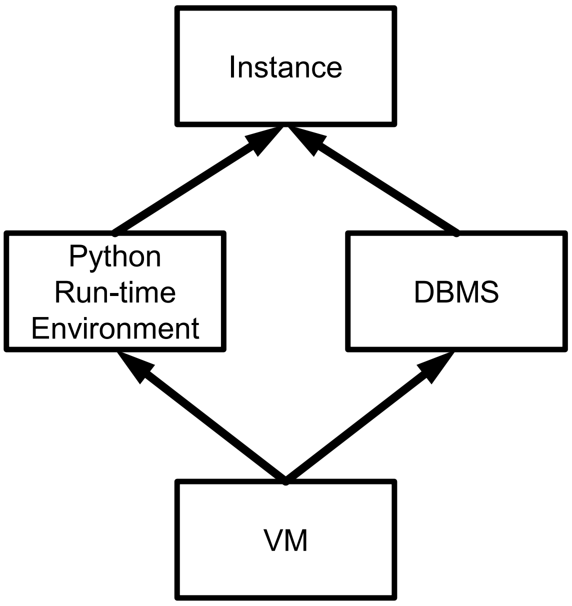

Figure 6.7.: Deployment Level of the investigated Scenario [Mor14]

ment Levels were created. Their structure is illustrated by Figure 6.7.

[2]This was analyzed by determining the required disc space of the Python source code.

Instance According to the definitions given in Chapter 4, the *instance* Deployment Level is the only mandatory Deployment Level. All Application Components need to be deployed on it. Thus, depending on the Application Component, Units are instantiated differently. For those Application Components that cover modules, Customers may express their Deployment Constraints about with whom they want to share instances of these modules. For the database Application Component, on the other hand, the instance Level allows Customers to express Deployment Constraints for sharing the database.

Database Management System The *database management system* Deployment Level is specific for the database Application Component. It is supposed to allow Customers to express their Deployment Constraints about with whom they are willing to share a database management system.

Python Run-time Environment Those Application Components that cover modules are running within an OpenERP Server. OpenERP, however, requires a Python run-time environment. Such an environment may host multiple instances of OpenERP. Thus, defining this Deployment Level allows Customers to express their Deployment Constraints about sharing the Python run-time environment.

Virtual Machine The virtual machine Deployment Level is the lowest Level and, thereby, hosts all other Levels. The idea of this Level is to have everything related to this scenario deployed on a dynamic infrastructure. Thus, it is possible to create the entire deployment automatically.

In subsection 4.2.1 requirements were discussed that Deployment Levels need to fulfill. All of the Deployment Levels just introduced, fulfill these requirements, with two exceptions. It has been discussed earlier that currently OpenERP implements Multi-Tenancy following the *separated databases* method. Thus, it is unfortunately necessary that all Customers use the *private*-Deployment Model on the instance Level of the database Component. Furthermore, as of now, OpenERP is not capable of handling more than one DBMS per installation. Thus, it is necessary that all Customers use the *public*-Deployment Model on the DBMS and the virtual machine Level of the database Component. This will limit the Customer's ability to express Deployment Constraints to modules only. This limitation will be addressed once again when the results of this chapter will be discussed (Section 6.5).

6.3.4 Creation of Example Scenario

Up to this point it has been discussed how OpenERP was altered in order to be component-based. Further, the Application Components and the Deployment Levels were defined.

Based on this, it is now possible to define a scenario in which Tenants use the component-based version of OpenERP following the Mixed-Tenancy approach.

The objective of this scenario is to encapsulate all expressible Deployment Constraints. Such a scenario has already been defined in Section 4.6.2. Thus, instead of recreating a new scenario, the following only describes how the scenario of Section 4.6.2 can be applied to the OpenERP Application Components and Deployment Levels.

Similar to the scenario of Subsection 4.6.2, six Tenants are created (T-A, T-B, T-C, T-D, T-E, T-F) for the case study. These Tenants express their constraints for all 27 Application Components on all relevant Deployment Levels. Again, the 26 Application Components that were created based on modules are deployed on the Deployment Levels instance, Python run-time environment, and a virtual machine. This corresponds very well to the Application Components AC-2, AC-3, and AC-4 of Section 4.6.2 since they were deployed on instance, application server, and virtual machine. Thus, for the scenario of this section the six Tenants' Deployment Constraints were applied to the 26 Application Components of this section. However, since there were only three Application Components in Section 4.6.2 but 26 here, all Deployment Constraints were reused multiple times. This way it is possible to ensure that all possible Deployment Constraints were expressed.

Based on this, there is only one previously defined OpenERP Application Component left. This 27th is the database being deployed on Deployment Level instance, database management system, and virtual machine. However, as discussed earlier, OpenERP implements Multi-Tenancy in the separated database approach[3]. Thus, for the six Tenants, it is only possible to express the Deployment Constraints that they all want a private Deployment Model on the instance Level and public for the Levels below.

Based on the Deployment Information that encapsulates all defined requirements, the top-down approach was used to create a Valid and Optimal Deployment and store it in a Deployment Configuration. This Deployment Configuration describes a total of 89 instances of Application Components and seven Python run-time environments. For the database, obviously, there are six instances and one Database Management System. Furthermore, for the entire scenario, a total of three virtual machines is required.

6.4 Realization of Deployment Platform for OpenERP

This section's purpose is to describe how a Mixed-Tenancy Deployment Platform for OpenERP may be realized. It does that by using the scenario, that was introduced in the previous section, as input. In Section 6.1 it has been analyzed that creating a Mixed-Tenancy Deployment Platform requires solving two problems. The first is that a Deployment automatically needs to be realized. The second is that communication between Application Components needs to be established. Thus, this section will discuss how the problems may be solved for OpenERP and, thereby, realize the Mixed-Tenancy Deployment Platform for OpenERP.

[3]The Separated Database approach was introduced in Section 3.1.1

In order to do that, this section is divided into two subsections. Subsection 6.4.1 discusses how a computed Deployment may be realized automatically. Subsection 6.4.2 introduces how connectors are implemented.

6.4.1 Problem 1: Automatic Deployment

Today, within a company that is Software Vendor and Operator at the same time, it is quite often the case that development and operations are in conflict with each other, even though they have the same mission to deliver valuable software to Customers. This is due to the fact that these two often act as distinct teams which results in suboptimal collaboration [Hüt12]. DevOps (a portmanteau of development and operations) is a method that is supposed to close that gap by stressing communication, collaboration and integration between software developers and operations [Hüt12].

This subsection will discuss how the first problem of creating a Mixed-Tenancy Deployment Platform may be tackled by applying techniques from the area of DevOps. Thus, it starts by introducing configuration management tools and choosing a useable one. This tool is then introduced and it is presented how it may be used to automatically deploy a Multi-Tenancy application based on a Deployment Configuration.

Selection of a Configuration Management Tool

One of the key concepts of DevOps is the automatic deployment and maintenance of a software's infrastructure. This is achieved through the documentation of these activities in executable code (commonly referred to as *Infrastructure as Code*). For this, configuration management tools provide domain specific languages that allow to create executable descriptions of management operations such as deployments [Hüt12]. Thus, using one of these tools seems to be a promising approach to tackle the challenge of automatically realizing a Deployment Configuration.

In [PTN13][pg. 28] a total of six configuration management tools were introduced. Of these six only two were considered relevant for this work since only those were able to be applied to multiple cloud environments. These are Chef [Ops13] and Puppet [Pup13].

Both tools have a similar feature set [Mor14] and both would have satisfied all requirements of this case study. However, Chef was chosen since Puppet's support for VMware vSphere is limited to the enterprise version of Puppet. The vSphere support was very important since it is part of the target environment of this case study. For Chef, on the other hand, VMware vSphere support is available through an extension for Chef's open source version. A more detailed analysis why Chef was chosen may be found in [Mor14].

Chef is a configuration management tool developed by a company called Opscode [Ops13]. It is available in two versions, one is open source, the other one is an enterprise version. Within this case study the open source version is used in version 11.6.0. The following are a few important terms related to Chef that are relevant for this work.

Cookbook A *cookbook* contains all the necessary entities for a specific scenario such as installing and configuring a specific application. A collection of widely used cookbooks is available for Chef. These are either maintained by Obscode or by the community around Chef. They include, for example, cookbooks for the installation and configuration of PostgreSQL. Each cookbook contains a collection of recipes dedicated to a specific scenario.

Recipe A *recipe* is a description of a management operation such as installing or configuring a specific application. It is created using the Ruby programming language.

Data Bag Data that is required for the execution of a recipe is stored in *data bags*. Such data may be everything that a particular operation needs (e.g. the address of a server where a database shall be set up). In fact, a data bag is a collection of data bag items that each store the data necessary for one execution of recipe. Data bag items are implemented using JSON as data format.

Knife *Knife* is a command line tool that is an alternative to Chef's web interface. It allows to manage the Chef server and local content such as cookbooks or data bags. Furthermore, an extension is available to be able to use knife in order to interact with VMWare vSphere. Using this extension, it is possible to create a virtual machine by simply calling knife with a set of parameters.

Automatic Deployment using Chef

Obviously, the input for the automatic deployment is the Deployment Configuration that is provided by the Deployment Configuration Generator. The Deployment Configuration has been realized as XML-file. XML was chosen since it allows easy exchange between different applications and may be created manually for testing purposes. A detailed description of the XML-File may be found in [Mor14].

As discussed in Section 6.1.1 the Deployment Configuration is read and processed by the Execution Engine. Since Chef shall be used as engine to actually execute the automatic deployment, the Chef server becomes part of the Execution Engine.

However, since the Deployment Configuration has a structure unknown to Chef, it is not possible for Chef to directly execute the Deployment Configuration. Thus, it is the job of the Execution Engine to translate the Deployment Configuration into data items on the Chef server. These data items are then accessed by the Chef server during the deployment process. For this, cookbooks were implemented each containing one recipe for every Deployment Level. Each one of them contains the description of operations necessary to instantiate this specific Deployment Level. The recipes are executed using the data items generated from the Deployment Configuration.

However, before this can be described, the first thing that needs to be done is to create a template virtual machine. This virtual machine is cloned for every virtual machine that is required by the Deployment Configuration. It was created using the VMWare vSphere-client application as a virtual machine using the operating system Ubuntu Linux 12.10 in its 64 bits server version. Furthermore, for the template virtual machine the options for Memory/CPU Hotplug were activated as this allows to add additional CPUs and memory while the virtual machine is running. In order to improve the time a cloning and deployment of a virtual machine takes, the template virtual machine was created with only five gigabytes of disk space. After the operating system was installed, the virtual machine was equipped with the VMWare tools which allow easier configuration through the management interface of the hypervisor. Furthermore, some DNS configuration was done in order to address cloned virtual machines by their host names and the entire OpenERP application was placed in a virtual machine. OpenERP was not installed but only placed there for future use. The reason for this is that once the virtual machine has been cloned, it will be necessary to instantiate those parts of OpenERP that are required according to the Deployment Configuration. If OpenERP is placed on the virtual machine already, it is not necessary to transfer it over the network, but it may be copied directly within the virtual machine.

Using the configuration management tool Chef it was possible to solve the automatic deployment problem very efficiently. This is due to the fact that Chef already provided a huge subset of the needed functionality. Furthermore, the usage of Chef has the advantage that even if setup procedures of, for example, the database change, it is not necessary to alter anything but getting new recipes for Chef. This problem has been solved in a very efficient and maintainable fashion.

6.4.2 Problem 2: Communication Mechanism

In Section 6.1.2 the Mixed-Tenancy connector pattern was introduced. It allows Application Component instances to communicate with each other. The following discusses how the pattern can be introduced to OpenERP.

OpenERP is equipped with an XML-RPC web service so that public methods of modules can be called remotely. It has been stated earlier that every Application Component encapsulates, in fact, its own instance of OpenERP. However, there is no built-in way for two instances of OpenERP to communicate via this service. An external application can call an OpenERP method through the web service, but every instance of OpenERP assumes that all installed modules are locally available. This way of communication is altered by the introduction of connectors. As discussed, a connector must be implemented to mimic a locally installed module. It then reroutes method calls by calling the respective method via XML-RPC on a remote instance. So the connector exposes the same interface as the original module and, thus, is a valid OpenERP module that can be loaded by the framework just like any other module. As only public methods can be called via XML-RPC, only those are being rerouted. To achieve interface compatibility between connector and original instance, all public methods in the connector must expose the same signature they expose in the original module. Additionally, all fields and private methods from the original module must be copied to the connector. This is necessary because some fields, that define the modules database structure, are evaluated against the actual database as the module is loaded. Also, some private methods may be required by other parts of the system. As they cannot be called remotely, they need to be copied to the connector. Finally, some additional private methods for packing and unpacking parameter data to submit via XML-RPC must be added, as well as a method called *_remote_call*. Instead of performing their original business logic, all public methods in the connector are implemented to call *_remote_call*, which in turn provides the necessary facilities to request the URL of the required remote instance from the EE via REST and then make the remote method call via XML-RPC.

Because of the number of OpenERP modules available and the fact that a connector is required for every module that is to be used in Mixed-Tenancy mode, manual implementation of the connectors is not feasible. Since all connectors share the interface with their original modules and the communication facility with the other connectors, however, automatic connector generation is the best way to go. Due to the fact that OpenERP is implemented using Python, the connector generator was implemented in Python as well. It can load OpenERP modules by importing the original module loading component of OpenERP. Once a module is loaded, the connector generator iterates over all submodules, classes and class members using Python reflection. All public methods are replaced with code in order to call the method *_remote_call* whereas all other parts of the module are left unchanged. Additionally, the method *_remote_call*, some other utility methods, and imports required by them are inserted at the beginning of the connector.

The connector generator can either generate a single connector or load all modules of an OpenERP installation and create connectors for all installed modules at once. Using the connector generator it was possible to minimize the effort necessary to create the connectors to a minimum. Further, if new versions of OpenERP are released, there should be very little effort necessary to create connectors for these as well.

6.5 Evaluation and Discussion of Results

In the previous sections, it has been presented how a Deployment Platform for OpenERP was created and OpenERP was migrated onto it. This section's purpose is to evaluate the presented solution and gain conclusions from this case study.

In order to do that this section is split into two parts. The first one evaluates the correctness of the introduction of Mixed-Tenancy to the component-based version of OpenERP. Based on this, conclusions related to resource utilization are discussed. Both parts include answering the related research questions (research question RQ-3.2 and research question RQ-3.2).

6.5.1 Evaluation of Mixed-Tenancy OpenERP

In the introduction of this chapter it was stated that the objective of this chapter is to investigate if it is possible to deploy an existing real-world application following the Mixed-Tenancy paradigm without altering functionality and behavior offered to the Customer. This leads to the conclusion that if a successful migration is achieved, the application's behavior and functionality is the same no matter if it has been deployed following the Mixed-Tenancy approach or the way it is meant to be deployed.

In the previous sections it has been discussed how the Mixed-Tenancy approach has been introduced to OpenERP. However, so far it has not been possible to provide evidence that this migration was actually successful. Thus, the success of the migration has been investigated. This was done by deploying two versions of OpenERP with identical test data and Tenants. The first version was deployed in the standard way according to the OpenERP documentation. The second version was deployed following the Mixed-Tenancy paradigm by realizing the scenario developed in Section 6.3.

These two versions were compared to each other based on defined test cases. Since the OpenERP system is huge and offers a lot of functionality, it was not possible to test the entire application in all its details. Instead, it was intended to test the entire application on a superficial level. This was done due to the assumption that differences in the behavior of the application may only be caused by the Deployment Platform and the connector pattern. Following this assumption, doing broad testing that involves all deployed Application Components would probably expose errors. Thus, about 150 test cases were defined and executed to cover the entire application with all its Application Components. In Appendix A.2 a full list of all executed test cases can be found.

During the test phase, the implementation of the connectors was improved by fixing errors discovered by the testers. Furthermore, at some point, it was noticed that a known bug in OpenERP ([Fay13]) did occur in the normal version, but not in the Mixed-Tenancy version. This was not expected and, thus, it was investigated. This investigation revealed that the known bug was caused by an error in the XML-RPC communication. Since this has been altered for the Mixed-Tenancy version, the bug did not occur in this version. A bug report was filed in the OpenERP bug tracking tool also including a patch [Rup13a]. After the development of the Deployment Platform was completed, both versions of the application behaved identical for all test cases.

To the extent of the test cases it is possible to say that the Mixed-Tenancy approach has successfully been introduced to the component-based version of OpenERP. Based on this, it is possible to conclude that in reality, there are cases where the Mixed-Tenancy approach may successfully be introduced, especially if applications' architecture is component-based.

6.5.2 Discussion of Resource Utilization

In the previous subsection it has been discussed that it was possible to apply the Mixed-Tenancy approach to OpenERP. However, in Section 6.3 it was analyzed that the original version of OpenERP is not a component-based application according to the definition of this work. Thus, it was discussed how OpenERP was split into Application Components that satisfy the requirements of this work. Unfortunately, this caused significant overhead and also caused the component-based version of OpenERP to require much more resources. In detail, a single instance of an OpenERP Application Component, created from modules, demands about 70 MB of RAM and 240 MB of disk space.

As stated earlier, in the investigated scenario there are a total of 89 instances of Application Components, 7 Python run-time environments, and 3 virtual machines required. This leads to a total resource demand of $6,998$ MB of RAM and $36,892$ MB of disk space to deploy the analyzed scenario (including Python run-time environments and virtual machines). These figures are also illustrated by Table 6.3. However, if these figures are compared to the upper bound (every Customer requires designated resources) and lower bound (all Customers share everything) that were introduced in Subsection 5.5.3, the results are as follows. The investigated scenario only requires about 54% of disc space and about 56% of RAM compared to resources required by the upper bound. Compared to the lower bound, the scenario requires about 326% more disc space and about 337% more RAM. A full overview of all figures, including the detailed figures for the individual Deployment Levels are illustrated by Table 6.3. This leads to the conclusion that resources may indeed be used more efficiently than in a Single-Tenancy deployment of the component-based OpenERP.

However, besides these discussions, another possible deployment that would satisfy all Deployment Constraints, is a designated deployment of 6 normal instances of the regular version of OpenERP – one per Tenant. This way every Tenant gets their own instance and, thus, all Deployment Constraints would be satisfied as well. This would require a total of $1,956$ MB of RAM and a total of $31,896$ MB of disc space. Compared to the deployment of the component-based version of OpenERP using the proposed scenario, this designated deployment requires only about 28% of RAM and 86% of disc space. As the connectors and Mixed-Tenancy Deployment Platform cause significant overhead with respect to run-time, it can further be assumed that deploying the original version of OpenERP this way also requires less CPU power.

All this leads to the conclusion that a component-based application that applies to the definitions of this work may indeed be used as a Mixed-Tenancy application. In fact, the Mixed-Tenancy approach may be introduced beneficially, meaning that it allows the Operator to save on operational cost. However, in such cases where the application is not component-based, altering it may lead to an application that does not allow the Operator to utilize resources more efficiently.

In such cases, however, it would still be an option to have Customers express their Deployment Constraints for the entire application. In such a scenario, the required number of instances of OpenERP would be lower than the number of

			Instance	**PRE**	**Virtual Machine**	**Σ**
Component-based OpenERP	All Private	∑ Instance	156	6	6	
		∑ Disk space	37,440	456	30,000	67,896
		∑ RAM	10,920	0	1,536	12,456
	Mixed Scenario	∑ Instance	89	7	3	
		∑ Disk space	21,360	532	15,000	36,892
		∑ RAM	6,230	0	768	6,998
	All Public	∑ Instance	26	1	1	
		∑ Disk space	6,240	76	5,000	11,316
		∑ RAM	1,820	0	256	2,076
OpenERP	All Private	∑ Instance	6	6	6	
		∑ Disk space	1,440	456	30,000	31,896
		∑ RAM	420	0	1,536	1,956
	Mixed Scenario	∑ Instance	-	-	-	
		∑ Disk space	-	-	-	
		∑ RAM	-	-	-	
	All Public	∑ Instance	1	1	1	
		∑ Disk space	240	76	5,000	5,316
		∑ RAM	70	0	256	326

PRE = Python Runtime Environment

Table 6.3.: Resource Demand for Mixed-Tenancy and Single-Tenancy scenario (Required storage in mega bytes)

Tenants if some Customers permit sharing with a community they trust. Further, there is no extra overhead in deploying OpenERP since it may be used as it was designed. Table 6.3 also stated the lower and upper bound for this scenario.

6.6 Summary

In this chapter the question was analyzed if the Mixed-Tenancy approach is applicable for real-world applications. By doing so, this chapter addressed research question RQ-3. In order to do that a case study was conducted that applied the Mixed-Tenancy approach to an existing Multi-Tenancy application.

The first step taken was to do a conceptual analysis of the challenges involved with creating a Deployment Platform. The following two challenges were identified and conceptual solutions were proposed.

Automatic Deployment The platform needs to be able to create a Mixed-Tenancy deployment of a Multi-Tenancy application according to a Deployment Configuration. The problem was analyzed and it was concluded that a Mixed-Tenancy Deployment needs to be provisioned bottom-up (starting with the lowest Deployment Level working its way up).

Communication Mechanism All instances of an Application Component of the same Tenants need to be able to communicate with each other. Since there may be multiple instances available of the same Application Component, the communication needs to be directed to this instance that serves the specific Customer that initiated the request. For this a behavioral pattern was created that allows to realize the communication by applying to the Deployment Constraints and not requiring to alter an existing application. The later point was achieved through separating platform logic from application logic.

These challenges need to be tackled specifically for every application for which a Mixed-Tenancy Deployment Platform shall be created.

However, the next step towards the analysis of applicability was to identify an application to which the Mixed-Tenancy approach shall be applied. The search for such an application was limited to open source ERP-systems that have a web-based user interface, are Multi-Tenant aware, and have a component-based architecture. After a detailed analysis, OpenERP was selected.

In a detailed analysis, it was concluded that OpenERP is in fact not component-based according to the definition of this work. Thus, a concept was presented of how Mixed-Tenancy may be introduced to OpenERP. This resulted in the creation of a new version of OpenERP that is Mixed-Tenancy enabled and is composed of 27 Application Components and has four Deployment Levels.

For these 27 Application Components a evaluative scenario was defined. In this scenario, there are six Tenants expressing their Deployment Constraints for all available Application Components and Deployment Levels. It was the goal of the scenario to cover all possible Deployment Constraints. Thus, the example that was used to perform the evaluation in Chapter 4 was used as basis.

As a next step, it was discussed how a Mixed-Tenancy Deployment Platform was developed for the Mixed-Tenancy version of OpenERP. Doing this included to solve the conceptual challenges that were identified. For the realization of the automatic deployment problem, Chef, a tool from the area of DevOps, was used. The connector pattern was implemented using Python. Furthermore, a connector generator was developed that allows to generate the required connectors automatically. This eliminates manual effort required for deploying a standard version of OpenERP following the Mixed-Tenancy paradigm.

Based on the Mixed-Tenancy Deployment Platform, it was possible to deploy the Mixed-Tenancy enabled version of OpenERP according to the defined scenario. Furthermore, based on about 150 test cases it was ensured that the application actually behaves just like a version that is deployed as it is meant to be deployed. This leads to the conclusion that it is possible to introduce the Mixed-Tenancy approach successfully to an application that is component-based according to the definition of this work.

With respect to resource utilization, it is possible to conclude that if the application is component-based, the Mixed-Tenancy approach allows the Operator to use resources more efficiently than following a Single-Tenancy model. If the application needs to be altered, however, this may cause significant overhead and, thereby, eliminate benefits for resource utilization.

In such cases, however, it would still be an option to have Customers express their Deployment Constraints for the entire application. In such a scenario, the required number of instances of OpenERP would be lower than the number of Tenants if some Customers permit sharing with a community they trust.

Chapter 7
Summary and Conclusion

7.1. Discussion of Results 163
7.1.1. Contributions 164
7.1.2. Limitations and Future Work 166
7.2. Summary 170

There is no real ending. It's just the place where you stop the story.

Frank Herbert

This chapter concludes this work by first discussing the contributions and limitations of this work. Starting from those, the opportunities for future research are being presented. The final section of this chapter will conclude this work by summarizing the overall thesis.

7.1 Discussion of Results

Chapter 2 analyzed the problem to be addressed by this work. From this analysis the general research question for this work was derived.

> *How can Customers' demand for expressing their Deployment Constraints and Operator's demand for having minimal infrastructure cost, both be satisfied at the same time?*

The next step was that more precise research questions were created to answer the general question. Those were first introduced in Subsection 2.2.3 and were repeated again for reference in the following discussions of contribution.

Research Question 1 (RQ-1) How can Customers' Deployment Constraints towards Mixed-Tenancy be described?

Research Question 1.1 (RQ-1.1) What shall be capturable in order to be able to describe Deployment Constraints?

Research Question 1.2 (RQ-1.2) How is the description being performed by the participating stakeholder?

Research Question 1.3 (RQ-1.3) What does a model look like that is capable of capturing Customers' Deployment Constraints?

Research Question 2 (RQ-2) How can a Valid and Optimal Deployment be found in a fast way?

Research Question 3 (RQ-3) Is it possible to apply the Mixed-Tenancy approach to existing composite Multi-Tenancy applications?

Research Question 3.1 (RQ-3.1) How can a platform realizing Mixed-Tenancy be created?

Research Question 3.2 (RQ-3.2) Does Mixed-Tenancy allow an Operator to use resources more efficiently as with Single-Tenancy?

The following subsections will discuss the contributions to those research questions and the limitations involved with the approach of this work.

7.1.1 Contributions

The novel contribution of this work is the Mixed-Tenancy approach. It is a hybrid approach between Single and Multi-Tenancy, which allows bridging the struggle between Customers that are hesitant to share resources with other Tenants and Operators that want to use resources as efficiently as possible. In order to realize the Mixed-Tenancy approach, the following are primary contributions produced by this work.

Description Model The Description Model allows Customers to express their Deployment Constraints. The objective of the Model was that it shall be generic in order to be applicable to a wide range of different applications. For the definition of Deployment Constraints, the Description Model provides four Deployments Models – Private, Public, White Hybrid, Black Hybrid, and Gray Hybrid. Using these Deployment Models it is possible for Customers to express if or with whom they are willing to share resources in a complex way by still allowing the Operator to keep its Customer base secret. This expression may be done for each Application Components on every applicable Deployment Level. The Description Model itself was defined completely and formally using first-order logic.

Deployment Computation Algorithm Within this work, two intuitive heuristics were introduced that allow approximating a Valid and Optimal Deployment. These are, the top-down approach that computes a Deployment starting from the highest Deployment Level down to the lowest, and the bottom-up approach that computes a Deployment starting from the lowest Deployment Level up to the highest. Based on a theoretical discussion, it was possible to prove that the top-down approach has a relative performance guarantee not better than 2 and the bottom-up approach has a relative performance guarantee not better than $3/2$. However, in addition, an experimental evaluation was performed that compared the two approaches' performance based on example problems. These experiments suggest preferring the top-down approach when computing a Deployment as it outperformed the bottom-up approach in most cases.

Mixed-Tenancy Deployment Platform This work gave a conceptual analysis of the challenges involved with creating a Mixed-Tenancy Deployment Platform. The two primary challenges are the automatic deployment according to a computed Deployment Configuration and the establishment of communication among the deployed Application Component Instances. A realization of a Deployment Platform highly depends on the specific composite Multi-Tenancy application that shall be provided to Customers. Within this work, OpenERP was identified as an example application and a Deployment Platform was realized. Based on the successful execution of about 150 test cases, it is possible to conclude that the realization was successful.

Based on these three primary results, it was possible to realize a working prototype that allows capturing Customers Deployment Constraints, approximating a Valid Deployment using the top-down approach and deploying OpenERP as a sample application according to the computed Deployment Configuration. Further, based on test cases, it was possible to determine that OpenERP worked properly and the Mixed-Tenancy approach was realized successfully.

In addition to the very high-level discussion of contributions, it is possible to give a more detailed description of contributions by discussing the results for the individual research questions. This is done in the following.

This work addressed research question RQ-1 by proposing a Description Model that has the primary objective to be as generic as possible. In response to research question RQ-1.1 an analysis was conducted and relevant entities were identified. In addition, the preliminary states of the model were continuously discussed with experts both from industry as well as from academia. Based on these discussions, the model was improved continuously. Finally, the model contained the following entities and their relationships to each other: Application Components, Deployment Levels, Tenants, Groups, and Dimensions. Based on these entities, it is possible for Operators and Customers to capture the application's structure and the Customers' Deployment Constraints. A more detailed description of how the model may be applied to a specific application was given by providing the description of a high-level process. This process is also the response to research question RQ-1.2. The model itself is the response to research question RQ-1.3. An incomplete representation of it is illustrated in Figure 4.6 using UML notation. Furthermore, Subsection 4.4 defines it formally.

In response to research question RQ-2, it was proven that the problem of computing a Valid and Optimal Deployment is NP-hard. This means there is no fast algorithm to compute a Valid and Optimal deployment unless $P = NP$. However, two heuristics were proposed that allow computing Valid and Optimal

Deployments. As already stated, they were evaluated both theoretically and experimentally.

In order to address research question RQ-3, a case study was conducted. The goal of this case study was to introduce the Mixed-Tenancy approach to a real-world open source ERP-System. After an analysis of available systems, OpenERP was selected. In response to research question RQ-3.1, the challenges involved with creating a Deployment Platform were discussed that allow introducing the Mixed-Tenancy approach to an existing Multi-Tenancy application without having to alter its source code. Based on this discussion, the platform has been realized for OpenERP. In order to validate that it is working properly, a total of about 150 test cases were defined and executed on a normal and a Mixed-Tenancy version of OpenERP. This allows concluding that it was possible to introduce the Mixed-Tenancy approach to an existing real-world application without having to alter the application's source code. Based on the investigated case, it is possible to respond to research question RQ-3.2 by stating that the Mixed-Tenancy approach allows using resources as efficiently as in a Single-Tenancy deployment if the application is component-based. If the application needs to be altered to fulfill this characteristic, however, this may cause significant overhead and, thereby, eliminates benefits for resource utilization. In such cases it would still be an option to have Customers express their Deployment Constraints for the entire application. This way it may be stated that there are cases where Mixed-Tenancy will allow Operators to use resources more efficiently.

7.1.2 Limitations and Future Work

In the previous section the contributions of this work were introduced. When they were created within this work, it was necessary, at some points, to propose assumptions in order to simplify the problems at hand. Obviously, these assumptions limit the contributions' applicability. The following mentions and introduces these limitations and discusses them. Further, each of these limitations bears the opportunity for future research. Thus, they will be addressed as well.

In Section 2.2 the scope of this work was defined. Part of this definition was the description of problems which are not to be addressed by this work. The following restates them briefly and discusses their impact on the contributions of this work and discusses opportunities for future research.

Changes over Time Within this work only the initial deployment of a Mixed-Tenancy application has been addressed. In reality, however, it is very likely that an initially provisioned deployment will have to be altered due to changing requirements (e.g. changing Customer base, changing Deployment Constraints). All challenges involved with doing that have not been addressed by this work and are still open for future research. The only exception is Appendix B of this work. It gives an initial analysis and solution approach for the problem of transforming a provisioned deployment into a new one while causing minimal cost. This appendix is based on [OY13], a Master's thesis that was created and supervised as part of this work.

Functional Variability In real-world, it is usually the case that different Customers have different requirements towards the same application. This work, however, assumed that a Mixed-Tenancy application provides the same features to all Customers. This assumption was made in order to be able to focus on the aspects specific to Mixed-Tenancy. Thus, it was possible to state that every Customer is always using instances of all Application Components – which decreases complexity. However, as discussed in Chapter 3 there is research conducted concerning the introduction of functional variability of Multi-Tenancy applications. Combining the conclusion's contributions created by this work with the techniques from related work is open for future research.

Collaboration It may be possible that the Mixed-Tenancy approach is used to enable collaboration between multiple Tenants by assigning them to the same instance of an Application Component and deactivating isolation. This would extend the approach to the possibility of allowing Customers to explicitly express with whom they want to share resources (instead of just naming the Tenants they would feel comfortable with) and a Deployment Computation Algorithm that is capable of realizing these Deployment Constraints. These questions have not been addressed by this work so far and, thus, are open for future research.

Deployment Variability In this work it was assumed that every Application Component requires one specific infrastructure stack. This assumption was made in order to decrease complexity. An alternative would be to allow that an Application Component has multiple alternative infrastructure stacks it may be deployed on. This opens up new opportunities for optimization since the number of different Units may be decreased. There has been research in this area (e.g. [FLM10]) connecting it to Mixed-Tenancy which, however, is still open for future research.

Implementation of Additional Security Measures The Mixed-Tenancy approach was motivated as an approach to decrease the risk of a Tenant's data being accessed by other Tenants. The basic idea was that if two Tenants do not share resources, it is possible to establish additional security mechanisms so separate their data. There are many such mechanisms possible (e.g. on a network level). However, they have not been explicitly analyzed within this work. The reason for this is that it is assumed that they are very specific to a given scenario. Thus, a detailed analysis of possible techniques to enforce Mixed-Tenancy is open for future research.

Other non-functional Requirements This work was solely focused on giving Customers the ability to express their Deployment Constraint about with which other Tenants they feel comfortable to share resources. It did not cover any other aspects like non-functional requirements such as the geographic location of the servers or guaranteed availability. There is related research that focuses on these aspects but combining it with the Mixed-Tenancy approach is still open for future research.

Pricing Models In Section 5.5.3 it has been discussed that it is in the interest of the Operator to provide incentives to Customers that allow sharing. Such incentives may come through a pricing model that promotes sharing by still allowing the Operator to earn a profit. The creation of such a model is out of scope for this work and, thus, an opportunity for future research.

Besides these general limitations and opportunities for future work, there are additional limitations for every research question. For research question RQ-1 the following limitations and opportunities for future research exist.

Validation Techniques for Customer Grouping One of the major shortcomings of the Description Model is that it requires the categorization of Tenants (e.g. associating all German Tenants to the German Group). If this data is not accurate and up to date, the entire approach suffers as Deployment Constraints expressed by Customer may not be met anymore. This work did not propose any techniques to ensure accuracy of data. Thus, this is up for future research. An interesting idea might be the creation of a collaborative model where Customers maintain the data jointly. Another approach would be to allow every Customer to maintain their view of the world and aggregate the information for Deployment Computation. Both ideas may actually promote the usage of technologies from the semantic web – which have already been used within this work.

Constraint Definition User Interface Within this work the Description Model was proposed for the capturing of Deployment Constraints. It is not, however, an intuitive way for Customers to express their Deployment Constraints. For this, it would be beneficial to have a special user interface. Such a user interface may actually generalize the Deployment Constraints to a more abstract level. It would be possible, for example, to provide the abstract Deployment Constraints high, medium and low security. These abstract Deployment Constraints may then be mapped to specific lower level Deployment Constraints like high to having everything private, low to having everything public and medium to sharing the virtual machine with everyone but the Application Component's instance only with companies from the Tenant's continent but not with competitors. For the case study conducted in Chapter 6, it was not possible to express for the database Application Component other Deployment Constraints than the Private one. Such a limitation may not be covered by the Description Model in its current state. However, it may be covered by the user interface and such mandatory constraints may be pre-filled.

With respect to research question RQ-2 this work has the following limitations.

Resource demand of Tenants When defining the General Mixed-Tenancy Deployment Problem, two types of resource demands were defined. Those caused by Tenants and those cause by the Units. This work was limited to optimization of the Resource Demand of Units since it was assumed that the Resource Demand of Tenants may be handled through virtualization technologies. However, this assumption is still open for validation, and if it should be proven to not be true, may cause a significant opportunity for future research.

Unlimited Resource Availability In the definition of the General Mixed-Tenancy Deployment Problem, it was assumed that any lower Level Unit may host an infinite number of Units on higher Levels. This assumption was made in order to decrease the complexity of the problem. However, in reality it may be wrong. Thus, addressing this issue is another opportunity for future research. An initial analysis of the problem and a first indication for a solution is given in Appendix C.

Deployment Computation Algorithm In this work two intuitive algorithmic approaches were proposed to approximate a Valid and Optimal Deployment. Further, it was proven that these approaches have a relative performance guarantee not better than 2 (top-down approach) and $3/2$ (bottom-up approach). Finding an algorithm with better performance is up to future research. Furthermore, it may prove useful to apply techniques from the area of artificial intelligence (e.g. simulated annealing [RN10]) to solve the problem. This is, however, up to future research as well.

Finally, for research question RQ-3 the following opportunities for future research have been identified.

Suitable Application The case study conducted in Chapter 6 was conducted based on OpenERP. As discussed, due to its architecture, OpenERP was not a very good application to be used in the case study. It is open for future research to find another, more suitable application that allows deploying individual Application Components independently from each other.

Communication Overhead Chapter 6 analyzed the problems of creating a Mixed-Tenancy Deployment Platform. One of the problems identified, was the development of a mechanism to establish communication among Application Component instances. Thus, the Mixed-Tenancy connector pattern was introduced. Within the description of the pattern, it was stated that for every communication the Execution Engine is requested to return the required destination. Since this is done every time, this causes overhead and increases run-time. Thus, it is up for future research to optimize this behavior. One possible way of doing that would be to store the requested information locally in the connectors and only update this information if the deployment changed. To come up with a smart way of managing this, the observer pattern ([GHJ94]) may be beneficial. It may allow extending the pattern the way that only minimal communication needs to be established.

7.2 Summary

This chapter summarized the entire work's results by outlining its contributions, limitations, and opportunities for future work.

Multi-Tenancy is an approach that promotes sharing of resources between multiple Tenants. On the one hand, this allows Operators to exploit economies of scale. On the other hand, the threat of data breaches makes Customers hesitant to use Multi-Tenancy applications. In order to satisfy both, the Operator and the Tenants, this work introduced the Mixed-Tenancy approach to allow Operators to utilize resources as efficiently as possible even if Customers express Deployment Constraints that are to be considered. This work addressed some of the major challenges involved with realizing the Mixed-Tenancy approach.

It proposed a Description Model that allows Customers to express their Deployment Constraints about if or with whom they are willing to share specific Application Components on specific Deployment Levels. This model was designed to be very generic in order to be reusable for a wide variety of composite Multi-Tenancy applications.

Based on all Deployment Constraints that are captured by the Description Model, it is possible to extract the aggregation information about which Tenants are allowed to share which Application Components on which Deployment Level. This information is called Deployment Information. It is the input for the Deployment Computation Algorithm, whose job is to compute a Valid and Optimal Deployment. A Deployment is Valid if it applies to all Deployment Constraints expressed by Customers. Furthermore, it is Valid and Optimal if it only utilizes minimal cost. Within this work it was proven that the problem of computing a Valid and Optimal deployment is NP-hard. Thus, it is not possible to do that in polynomial time. However, two intuitive approaches were introduced – top-down and bottom-up. They were analyzed both theoretically and experimentally. The theoretical analysis revealed that they only have a relative performance guarantee not better than 2 and $3/2$. The experimental analysis revealed that in most cases the top-down approach outperforms the bottom-up approach. However, due of the complexity of the problem, it was not possible to evaluate the overall quality of both approaches compared to an optimal solution.

The final step of the work was to evaluate the applicability of the Mixed-Tenancy approach to existing real-world applications. This was done by performing a case study where the approach was introduced to OpenERP, an open source ERP-System. Based on this case study, it is possible to conclude that there are cases where the Mixed-Tenancy approach may successfully be applied in real-world.

Furthermore, even though the approach has only been applied to applications, it is possible to apply the approach also to systems that are used to deliver the other cloud computing service models Infrastructure-as-a-Service and Platform-as-a-Service (e.g. Customers may choose with which other Customers their virtual machines would be allowed to use the same hypervisor). Due to the lower complexity of these cases, both the Description Model and the Deployment Computation Algorithm may be utilized without having to alter them.

Appendix A
Details related to Case Study

A.1. Description of Application Components 171
A.2. Evaluated Test Cases 178

Not everything that can be counted counts and not everything that counts can be counted.

Albert Einstein

This appendix's purpose is to provide additional details for the case study conducted in Chapter 6. In Section A.1 a detailed description of the created Application Components may be found. Further, the test cases that were used for the evaluation may be found in Section A.2.

As already stated in the introduction of Chapter 6, the case study was conducted in cooperation with Matthias Reinhardt, Malte Rupprecht, Björn Morr, and Brian Korduan.

A.1 Description of Application Components

It is the purpose of this section to give an overview of the 26 Application Components that were created within the case study of this work. It was stated in 6.3.2 that Application Components were created based on the OpenERP apps the standard version comes with. Thus, they are listed in the following style.

{technical app name} ({app name visible to Customers}) { brief description of provided functionality based on [Ope13]} *{(contains xx modules)}* {technical name of module 1}, {technical name of module 2}, <u>{technical name of app}</u>, {technical name of module 3}

The following is the complete list of the created Application Components. Description text of apps are direct quotes from the description of apps that are part of the OpenERP documentation [Ope14].

account (eInvoicing) Accounting and Financial Management. Financial and accounting module that covers: * General Accounting * Cost/Analytic accounting * Third party accounting * Taxes management * Budgets * Customer and Supplier Invoices * Bank statements * Reconciliation process by partner Creates a dashboard for accountants that includes: * List of Customer Invoice to Approve * Company Analysis * Graph of Treasury The processes like maintaining of general ledger is done through the defined financial Journals (entry move line orgrouping is maintained through journal) for a particular financial year and for preparation of vouchers there is a module named account_voucher. *(Contains 7 modules)*
account, base_setup, product, analytic, process, board, edi

account_accountant (Accounting and Finance) Accounting Access Rights It gives the Administrator user access to all accounting features such as journal items and the chart of accounts. It assigns manager and user access rights to the Administrator and only user rights to the Demo user. *(Contains 2 modules)*
account_accountant, account_voucher

account_voucher (eInvoicing & Payments) Invoicing and Payments by Accounting Voucher and Receipts The specific and easytouse Invoicing system in OpenERP allows you to keep track of your accounting, even when you are not an accountant. It provides an easy way to follow up on your suppliers and Customers. You could use this simplified accounting in case you work with an (external) account to keep your books, and you still want to keep track of payments. The Invoicing system includes receipts and vouchers (an easy way to keep track of sales and purchases). It also offers you an easy method of registering payments, without having to encode complete abstracts of account. This module manages: * Voucher Entry * Voucher Receipt [Sales & Purchase] * Voucher Payment [Customer & Supplier] *(Contains 2 modules)*
account_voucher, account

sale (Sales Management) Manage sales quotations and orders This application allows you to manage your sales goals in an effective and efficient manner by keeping track of all sales orders and history. It handles the full sales workflow: * **Quotation** > **Sales order** > **Invoice** Preferences (only with Warehouse Management installed) If you also installed the Warehouse Management, you can deal with the following preferences: * Shipping: Choice of delivery at once or partial delivery * Invoicing: choose how invoices will be paid * Incoterms: International Commercial terms You can choose flexible invoicing methods: * *On Demand*: Invoices are created manually from Sales Orders when needed * *On Delivery Order*: Invoices are generated from picking (delivery) * *Before Delivery*: A Draft invoice is created and must be paid before delivery The Dashboard for the Sales Manager will include * My Quotations * Monthly Turnover (Graph)

(Contains 2 modules)
sale, account_voucher

purchase (Purchase Management) Manage goods requirement by Purchase Orders easily Purchase management enables you to track your suppliers price quotations and convert them into purchase orders if necessary. OpenERP has several methods of monitoring invoices and tracking the receipt of ordered goods. You can handle partial deliveries in OpenERP, so you can keep track of items that are still to be delivered in your orders, and you can issue reminders automatically. OpenERP's replenishment management rules enable the system to generate draft purchase orders automatically, or you can configure it to run a lean process driven entirely by current production needs. Dashboard / Reports for Purchase Management will include: * Request for Quotations * Purchase Orders Waiting Approval * Monthly Purchases by Category * Receptions Analysis * Purchase Analysis *(Contains 4 modules)*
purchase, procurement, process, stock

mail (Social Network) Business oriented Social Networking The Social Networking module provides a unified social network abstraction layer allowing applications to display a complete communication history on documents with a fullyintegrated email and message management system. It enables the users to read and send messages as well as emails. It also provides a feeds page combined to a subscription mechanism that allows to follow documents and to be constantly updated about recent news. Main Features * Clean and renewed communication history for any OpenERP document that can act as a discussion topic * Subscription mechanism to be updated about new messages on interesting documents * Unified feeds page to see recent messages and activity on followed documents * User communication through the feeds page * Threaded discussion design on documents * Relies on the global outgoing mail server an integrated email management system allowing to send emails with a configurable schedulerbased processing engine * Includes an extensible generic email composition assistant, that can turn into a massmailing assistant and is capable of interpreting simple *placeholder expressions* that will be replaced with dynamic data when each email is actually sent. *(Contains 3 modules)*
mail, base, base_setup

hr_expense (Expense Management) Manage expenses by Employees This application allows you to manage your employees daily expenses. It gives you access to your employees' fee notes and give you the right to complete and validate or refuse the notes. After validation it creates an invoice for the employee. Employee can encode their own expenses and the validation flow puts it automatically in the accounting after validation by managers. The whole flow is implemented as: * Draft expense * Confirmation of the sheet by the employee * Validation by his manager * Validation by the accountant and receipt creation This module also uses analytic accounting

and is compatible with the invoice on timesheet module so that you are able to automatically reinvoice your Customers expenses if your work by project. *(Contains 4 modules)*
hr_expense, account_accountant, hr, account_voucher

hr_timesheet_sheet (Timesheets) Record and validate timesheets and attendances easily This application supplies a new screen enabling you to manage both attendances (Sign in/Sign out) and your work encoding (timesheet) by period. Timesheet entries are made by employees each day. At the end of the defined period, employees validate their sheet and the manager must then approve his teams entries. Periods are defined in the company forms and you can set them to run monthly or weekly. The complete timesheet validation process is: * Draft sheet * Confirmation at the end of the period by the employee * Validation by the project manager The validation can be configured in the company: * Period size (Day, Week, Month) * Maximal difference between timesheet and attendances *(Contains 4 modules)*
hr_timesheet_sheet, process, hr_timesheet_invoice, hr_timesheet

event (Events Organisation) Organization and management of Events. The event module allows you to efficiently organise events and all related tasks: planification, registration tracking, attendances, etc. Key Features * Manage your Events and Registrations * Use emails to automatically confirm and send acknowledgements for any event registration *(Contains 4 modules)*
event, board, base_setup, email_template

base_calendar (Calendar) This is a fullfeatured calendar system. It supports: Calendar of events Recurring events If you need to manage your meetings, you should install the CRM module. *(Contains 5 modules)*
base_calendar, base, mail, base_action_rule, base_status

stock (Warehouse Management) Manage multiwarehouses, multi and structured stock locations The warehouse and inventory management is based on a hierarchical location structure, from warehouses to storage bins. The double entry inventory system allows you to manage Customers, suppliers as well as manufacturing inventories. OpenERP has the capacity to manage lots and serial numbers ensuring compliance with the traceability requirements imposed by the majority of industries. Key Features * Moves history and planning, * Stock valuation (standard or average price, ...) * Robustness faced with Inventory differences * Automatic reordering rules * Support for barcodes * Rapid detection of mistakes through double entry system * Traceability (Upstream / Downstream, Serial numbers, ...) Dashboard / Reports for Warehouse Management will include: * Incoming Products (Graph) * Outgoing Products (Graph) * Procurement in Exception * Inventory Analysis * Last Product Inventories * Moves Analysis *(Contains 3 modules)*
stock, product, account

contacts (Address Book) This module gives you a quick view of your address book, accessible from your home page. You can track your suppliers, Cus-

tomers and other contacts. *(Contains 2 modules)*
contacts, mail

crm (CRM) The generic OpenERP Customer Relationship Management This application enables a group of people to intelligently and efficiently manage leads, opportunities, meetings and phone calls. It manages key tasks such as communication, identification, prioritization, assignment, resolution and notification. OpenERP ensures that all cases are successfully tracked by users, Customers and suppliers. It can automatically send reminders, escalate the request, trigger specific methods and many other actions based on your own enterprise rules. The greatest thing about this system is that users dont need to do anything special. The CRM module has an email gateway for the synchronization interface between mails and OpenERP. That way, users can just send emails to the request tracker. OpenERP will take care of thanking them for their message, automatically routing it to the appropriate staff and make sure all future correspondence gets to the right place. Dashboard for CRM will include: * Planned Revenue by Stage and User (graph) * Opportunities by Stage (graph) *(Contains 11 modules)*
crm, base_setup, process, mail, base_action_rule, email_template, resource, base_calendar, board, base_status, fetchmail

fleet (Fleet Management) Vehicle, leasing, insurances, cost With this module, OpenERP helps you managing all your vehicles, the contracts associated to those vehicle as well as services, fuel log entries, costs and many other features necessary to the management of your fleet of vehicle(s) Main Features * Add vehicles to your fleet * Manage contracts for vehicles * Reminder when a contract reach its expiration date * Add services, fuel log entry, odometer values for all vehicles * Show all costs associated to a vehicle or to a type of service * Analysis graph for costs *(Contains 1 modules)*
fleet

hr (Employee Directory) Human Resources Management This application enables you to manage important aspects of your companys staff and other details such as their skills, contacts, working time... You can manage: * Employees and hierarchies : You can define your employee with User and display hierarchies * HR Departments * HR Jobs *(Contains 5 modules)*
hr, resource, base_setup, mail, board

hr_evaluation (Employee Appraisals) Periodical Employees evaluation and appraisals By using this application you can maintain the motivational process by doing periodical evaluations of your employees performance. The regular assessment of human resources can benefit your people as well your organization. An evaluation plan can be assigned to each employee. These plans define the frequency and the way you manage your periodic personal evaluations. You will be able to define steps and attach interview forms to each step. Manages several types of evaluations: bottomup, topdown,

selfevaluations and the final evaluation by the manager. Key Features * Ability to create employees evaluations. * An evaluation can be created by an employee for subordinates, juniors as well as his manager. * The evaluation is done according to a plan in which various surveys can be created. Each survey can be answered by a particular level in the employees hierarchy. The final review and evaluation is done by the manager. * Every evaluation filled by employees can be viewed in a PDF form. * Interview Requests are generated automatically by OpenERP according to employees evaluation plans. Each user receives automatic emails and requests to perform a periodical evaluation of their colleagues. *(Contains 4 modules)*
hr_evaluation, base_calendar, hr, survey

hr_holidays (Leave Management) Manage leaves and allocation requests. This application controls the holiday schedule of your company. It allows employees to request holidays. Then, managers can review requests for holidays and approve or reject them. This way you can control the overall holiday planning for the company or department. You can configure several kinds of leaves (sickness, holidays, paid days, ...) and allocate leaves to an employee or department quickly using allocation requests. An employee can also make a request for more days off by making a new Allocation. It will increase the total of available days for that leave type (if the request is accepted). You can keep track of leaves in different ways by following reports: * Leaves Summary * Leaves by Department * Leaves Analysis A synchronization with an internal agenda (Meetings of the CRM module) is also possible in order to automatically create a meeting when a holiday request is accepted by setting up a type of meeting in Leave Type. *(Contains 5 modules)*
hr_holidays, resource, hr, process, base_calendar

hr_recruitment (Recruitment Process) Manage job positions and the recruitment process This application allows you to easily keep track of jobs, vacancies, applications, interviews... It is integrated with the mail gateway to automatically fetch email sent to <jobs@yourcompany.com> in the list of applications. Its also integrated with the document management system to store and search in the CV base and find the candidate that you are looking for. Similarly, it is integrated with the survey module to allow you to define interviews for different jobs. You can define the different phases of interviews and easily rate the applicant from the kanban view. *(Contains 7 modules)*
hr_recruitment, hr, survey, decimal_precision, base_calendar, base_status, fetchmail

lunch (Lunch Orders) The base module to manage lunch. Many companies order sandwiches, pizzas and other, from usual suppliers, for their employees to offer them more facilities. However lunches management within the company requires proper administration especially when the number of employees or suppliers is important. The “Lunch Order” module has been

developed to make this management easier but also to offer employees more tools and usability. In addition to a full meal and supplier management, this module offers the possibility to display warning and provides quick order selection based on employee's preferences. If you want to save your employees time and avoid them to always have coins in their pockets, this module is essential. *(Contains 2 modules)*
lunch, base

mrp (MRP) Manage the Manufacturing process in OpenERP The manufacturing module allows you to cover planning, ordering, stocks and the manufacturing or assembly of products from raw materials and components. It handles the consumption and production of products according to a bill of materials and the necessary operations on machinery, tools or human resources according to routings. It supports complete integration and planification of stockable goods, consumables or services. Services are completely integrated with the rest of the software. For instance, you can set up a subcontracting service in a bill of materials to automatically purchase on order the assembly of your production. Key Features * Make to Stock/Make to Order * Multilevel bill of materials, no limit * Multilevel routing, no limit * Routing and work center integrated with analytic accounting * Periodical scheduler computation * Allows to browse bills of materials in a complete structure that includes child and phantom bills of materials Dashboard / Reports for MRP will include: * Procurements in Exception (Graph) * Stock Value Variation (Graph) * Work Order Analysis *(Contains 7 modules)*
mrp, resource, product, process, purchase, stock, procurement

procurement (Procurements) This is the module for computing Procurements. In the MRP process, procurements orders are created to launch manufacturing orders, purchase orders, stock allocations. Procurement orders are generated automatically by the system and unless there is a problem, the user will not be notified. In case of problems, the system will raise some procurement exceptions to inform the user about blocking problems that need to be resolved manually (like, missing BoM structure or missing supplier). The procurement order will schedule a proposal for automatic procurement for the product which needs replenishment. This procurement will start a task, either a purchase order form for the supplier, or a production order depending on the products configuration. *(Contains 5 modules)*
procurement, base, process, product, stock

project (Project Management) Track multilevel projects, tasks, work done on tasks This application allows an operational project management system to organize your activities into tasks and plan the work you need to get the tasks completed. Gantt diagrams will give you a graphical representation of your project plans, as well as resources availability and workload. Dashboard / Reports for Project Management will include: * My Tasks * Open Tasks * Tasks Analysis * Cumulative Flow *(Contains 9 modules)* project, base_setup, product, analytic, mail, web_kanban, resource, board, base_status

note (Notes) This module allows users to create their own notes inside OpenERP Use notes to write meeting minutes, organize ideas, organize personnal todo lists, etc. Each user manages his own personnal Notes. Notes are available to their authors only, but they can share notes to others users so that several people can work on the same note in real time. Its very efficient to share meeting minutes. Notes can be found in the Home menu. *(Contains 2 modules)*
note, mail

point_of_sale (Point of Sale) Quick and Easy sale process This module allows you to manage your shop sales very easily with a fully web based touchscreen interface. It is compatible with all PC tablets and the iPad, offering multiple payment methods. Product selection can be done in several ways: * Using a barcode reader * Browsing through categories of products or via a text search. Main Features * Fast encoding of the sale * Choose one payment method (the quick way) or split the payment between several payment methods * Computation of the amount of money to return * Create and confirm the picking list automatically * Allows the user to create an invoice automatically * Refund previous sales *(Contains 2 modules)*
point_of_sale, sale_stock

portal (Portal) Customize access to your OpenERP database to external users by creating portals. A portal defines a specific user menu and access rights for its members. This menu can ben seen by portal members, anonymous users and any other user that have the access to technical features (e.g. the administrator). Also, each portal member is linked to a specific partner. The module also associates user groups to the portal users (adding a group in the portal automatically adds it to the portal users, etc). That feature is very handy when used in combination with the module share. *(Contains 4 modules)*
portal, base, share, auth_signup

project_issue (Issue Tracker) Track Issues/Bugs Management for Projects This application allows you to manage the issues you might face in a project like bugs in a system, client complaints or material breakdowns. It allows the manager to quickly check the issues, assign them and decide on their status quickly as they evolve. *(Contains 4 modules)*
project_issue, base_status, crm, project

A.2 Evaluated Test Cases

The following lists the test cases that were created to evaluate if it was possible to apply the Mixed-Tenancy approach to OpenERP successfully.

ID	Description	Steps
TCMMI0001	read a inbox message	1. left click module Messaging 2. left click Inbox 3. left click on a message
TCMMI0002	compose a new message	1. left click module Messaging 2. left click Inbox 3. left click compose a new message 4. fill in Recipients(hans@hans.de) 5. click create hans@hans.de 6. fill in subject :hans 7. fill in message :hans 8. left click :send
TCMMTM0001	open to:me messages	1. left click module Messaging 2. left click to: me
TCMMTM0002	mark a to: me message as to: do	1. left click module Messaging 2. left click to: me 3. left click at Mark as todo (Button seems like a favorite star)
TCMMTD0001	open to-do list	1. left click module Messaging 2. left click to: do
TCMMTD0002	completing a to-do task	1. left click module Messaging 2. left click to: do 3. left click done (checkmark on the right side)
TCMMTD0003	repeating TCM-0006: completing a to-do task	1. left click module Messaging 2. left click to: do 3. left click done (checkmark on the right side)
TCMOCA0001	open calender	1. left click module Messaging 2. left click Calender
TCMOCA0002	create meeting in calender	1. left click module Messaging 2. left click Calender - go to list view 3. left click Create 4. fill in Meeting Subject :Betriebsrat13&14 5. fill in attendees :hans@hans.de 6. set starting at 11/13/2013 10:00:14 7. create and set the tag :Betriebsrat 8. fill in location :Darmstadt 9. fill in duration :2 10. fill in Description :Hallo 11&13 11. left click save
TCMONO0001	open notes	1. left click module Messaging 2.left click notes
TCMONO0002	create a new note	1. left click module Messaging 2.left click notes 3. left click create 4. fill in tag :Hans 5. fill in text BlaBla 6. save
TCMMGJAG0001	open join a group	1. left click Messaging 2. left click join a group

ID	Description	Steps
TCSSCU0001	open Customers	1. left click module Sales 2. left click Customers
TCSSCU0002	create Customer	1. left click module Sales 2. left click Customers 3. left click create 4. fill in Name:Peter 5. left click save
TCSSLE0001	open leads	1. left click module Sales 2. left click leads
TCSSLE0002	read a lead	1. left click module Sales 2. left click leads 3. select a lead
TCSSLE0003	convert lead to oppertunity	1. left click module Sales 2. left click leads 3. select a lead 4. left click convert to opportunity 5. select Conversion Action :Convert to opportunity 6. select Related Customer :Create a new Customer 7. left click Create Opportunity
TCSSQU0001	select Quotations	1. left click module Sales 2. left click Quotations
TCSSQU0002	create a Quotation	1. left click module Sales 2. left click Quotations 3. left click create 4. select Customer:Agrolait 5. select Contract/Analytic :Your Company/Internal/Administrative 6. left click save
TCSSSO0001	select Sales Orders	1. left click module Sales 2. left click Sales Orders
TCSSSO0002	cancel an order	1. left click module Sales 2. left click Sales Orders 3. select an order 4. left click Cancel Order
TCSSCO0001	create a contract	1. left click module Sales 2. left click contracts 3. click create 4. fill in datas 5. click save
TCSASSCL0001	create a claim	1. left click module Sales 2. left click Claims 3. click create 4. fill in datas 5. click save
TCSASSCL0002	settle a claim	1. left click module Sales 2. left click Claims 3. select Claim 4. click Settle

ID	Description	Steps
TCSASSCL0003	Reject a claim	1. left click module Sales 2. left click Claims 3. select Claim 4. click Reject
TCSICTR0001	create a contract to renew	1. left click module Sales 2. left click contracts to renew 3. click create 4. fill in datas 5. click save
TCWRDBOIS0001	create a incomming Shipment	1. left click module Warehouse 2. select Incomming Shipments 3. left click create 4- fill in test datas 5. left click save
TCWRDBOIS0002	confirm and receive a incomming Shipment	1. left click module Warehouse 2. select Incomming Shipments 3.choose a incomming Shipment 4. left click confirm and receive
TCWRDBODO0001	create a delivery order	1. left click module Warehouse 2. select Delivery Orders 3. left click create 4. fill in test datas 5. click save
TCWRDBODO0002	force availability of a delivery order	1. left click module Warehouse 2. select Delivery Orders 3. choose a delivery order 4. left click Force Availability
TCWRDPIP0001	share incomming products	1. left click module Warehouse 2. select incomming products 3. mark incomming products 4. left click more 5. left click share 6. choose Email as sharing option 7. fill in text 8. left click send
TCWRDPDP0001	create deliver product	1. left click module Warehouse 2. select Deliver Products 3. left click create 4. fill in test datas 5. click save 6. click process later
TCWICPI0001	create a physical inventory	1. left click module Warehouse 2. select physical inventory 3. left click create 4. left click save 5. left click confirm inventory

ID	Description	Steps
TCWSPE0001	compute stock minimum rules	1. left click module Warehouse 2. select Procurement Exceptions 3. mark products 4. left click more 5. select Compute Stock minimum Rules 6. click Compute Stock in popup window
TCWPP0001	create a product	1. left click module Warehouse 2. select Products 3. left click create 4. fill in Testdatas 5. left click save
TCWPP0002	print created product	1. after step 4 of TCW-0010 select print 2. click product labels
TCEEOE0001	create an event	1. left click module Events 2. select Events 3. left click create 4. fill in datas 5. left click save
TCEEOR0001	confirm a registration	1. left click module Events 2. select Registrations 3. choose and select a registration 4.left click confirm
TCRDMD0001	select My Dashboard	1. left click module Reporting 2. select My Dashboard
TCRDC0001	select CRM	1. left click module Reporting 2. select CRM
TCRDS0001	select Sales	1. left click module Reporting 2. select Sales
TCRDPU0001	select Purchases	1. left click module Reporting 2. select Purchases
TCRDW0001	select Warehouse	1. left click module Reporting 2. select Warehouse
TCRDM0001	select Manufacturing	1. left click module Reporting 2. select Manufacturing
TCRDPR0001	select Project	1. left click module Reporting 2. select Project
TCRDE0001	select Events	1. left click module Reporting 2. select Events
TCRDA0001	select Accounting	1. left click module Reporting 2. select Accounting
TCRDF0001	select Fleet	1. left click module Reporting 2. select Fleet
TCRDHR0001	select Human Resources	1. left click module Reporting 2. select Human Resources
TCRSUPS0001	select Print Surveys	1. left click module Reporting 2. select Print Surveys

ID	Description	Steps
TCRSUSS0001	select Surveys Statistics	1. left click module Reporting 2. select Surveys Statistic
TCRSUSA0001	select Surveys Answers	1. left click module Reporting 2. select Surveys Answers
TCRSUBA0001	select Browse Answers	1. left click module Reporting 2. select Browse Answer
TCRSLA0001	select Leads Analysis	1. left click module Reporting 2. select Leads Analysis
TCRSOA0001	select Opportunities Analysis	1. left click module Reporting 2. select Opportunities Analysis
TCRSSA0001	select Sales Analysis	1. left click module Reporting 2. select Sales Analysis
TCRSPCA0001	select Phone Calls Analysis	1. left click module Reporting 2. select Phone Calls Analysis
TCRPUPA0001	select Purchase Analysis	1. left click module Reporting 2. select Purchases Analysis
TCRWRA0001	select Receptions Analysis	1. left click module Reporting 2. select Receptions Analysis
TCRWLPI0001	select Last Product Inventories	1. left click module Reporting 2. select Product Inventories
TCRWMA0001	select Moves Analysis	1. left click module Reporting 2. select Moves Analysis
TCRWIA0001	select Inventory Analysis	1. left click module Reporting 2. select Inventory Analysis
TCRPRTA0001	select Tasks Analysis	1. left click module Reporting 2. select Tasts Analysis
TCRPRCF0001	select Cumulative Flow	1. left click module Reporting 2. select Cumulative Flow
TCRPRCA0001	select Claims Analysis	1. left click module Reporting 2. select Claims Analysis
TCRPRIA0001	select Issues Analysis	1. left click module Reporting 2. select Issues Analysis
TCREEA0001	select Events Analysis	1. left click module Reporting 2. select Events Analysis
TCRAIA0001	select Invoices Analysis	1. left click module Reporting 2. select Invoices Analysis
TCRAEA0001	select Entries Analysis	1. left click module Reporting 2. select Entries Analysis
TCRATA0001	select Treasury Analysis	1. left click module Reporting 2. select Teasury Analsyis
TCRASRA0001	select Sales Receipts Analysis	1. left click module Reporting 2. select Sales Receipts Analysis
TCRAAEA0001	select Analytic Entries Analysis	1. left click module Reporting 2. select Analytic Entries Analysis
TCRHRRA0001	select Recruitment Analysis	1. left click module Reporting 2. select Recruitment Analysis

ID	Description	Steps
TCRHREA0001	select Expenses Analysis	1. left click module Reporting 2. select Expenses Analysis
TCRHRAA0001	select Appraisal Analysis	1. left click module Reporting 2. select Appraisal Analysis
TCRHRET0001	select Employee Timesheet	1. left click module Reporting 2. select Employee Timesheet
TCRHRTA0001	select Timesheet Analysis	1. left click module Reporting 2. select Timesheet Analysis
TCRHRTSA0001	select Timesheet Sheet Analysis	1. left click module Reporting 2. select Timesheet Sheet Analysis
TCRHRLA0001	select Leaves Analysis	1. left click module Reporting 2. select Leaves Analysis
TCRHRR0001	select Reports	1. left click module Reporting 2. select Reports
TCRFCA0001	select Costs Analysis	1. left click module Reporting 2. select Costs Analysis
TCRFICA0001	select Indicative Costs Analysis	1. left click module Reporting 2. select Indicative Costs Analysis
TCRPOSOA0001	select Orders Analysis	1. left click module Reporting 2. select Orders Analysis
TCRPOSSD0001	select Sale Details	1. left click module Reporting 2. select Sale Details
TCMAMMO0001	creat a manufacturing order	1. left click module Manufacturing 2. select Manufacturing Orders 3. left click create 4. left click save 5. left click Confirm Production
TCMAPBOM0001	share bills of materials	1. left click module Manufacturing 2. select Bill of Materials 3. mark some bills 4. left click more 5. left click share 6. choose share method Email 7. fill in text 8. left click share
TCLLNO0001	order a new meal	1. left click module Lunch 2. select New Order 3. choose a produkt 4. save orders
TCLLYLA0001	create a new lunch account	1. left click module Lunch 2. select Your Lunch Account 3. Left click Create
TCLCP0001	create meal	1. left click module Lunch 2. select Products 3. Left click Create 4. fill in datas 5. left click save

ID	Description	Steps
TCLCPC0001	create product categories	1. left click module Lunch 2. select Product Categories 3. Left click Create 4. fill in datas 5. left click save
TCACCI0001	select Customer invoices	1. left click select module:Accounting 2. choose a Customer invoice
TCACCI0002	validate a Customer invoice	1. left click select module:Accounting 2. choose a Customer invoice 3. left click validate
TCACCR0001	create a Customer refund	1. select module : Accounting 2. select Customer Refunds 3. click create 4. fill in datas 5. click save
TCACC0001	select Customer	1. left click select module:Accounting 2. select Customers
TCACC0002	create a new Customer	1. left click select module:Accounting 2. select Customers 3. left click create 4. fill in name :Hans 5. left click save
TCASSR0001	Validate supplier refund	1. left click module Accounting 2. select Supplier Refund 3. choose a supplier refund 4. left click validate
TCASPR0001	create purchase receipts	1. left click module Accounting 2. select Purchase Receipts 3. left click create 4. fill in fields 5. save
TCASPR0002	validate purchase receipts	1. left click module Accounting 2. select Purchase Receipts 3. choose a purchase receipt 4. click validate
TCASSP0001	create a supplier payment	1. left click module Accounting 2. select supplier payment 3. fill in datas 4. click save
TCABACBS0001	create a bank statement	1. left click module Accounting 2. select Bank Statements 3. click create 4. fill in datas 5. click save
TCAJEJE0001	create journal entries	1. left click module Accounting 2. select Journal Entries 3. left click create 4. fill in fields 5. left click save

ID	Description	Steps
TCACCOA0001	open a charts of account	1.open module :Accounting 2.select Charts of Accounts 3. choose fiscal year 4. click open Charts
TCAPPRE0001	create recurring entries	1. select module Accounting 2. select Recurring Entries 3. select Define Recurring Entries 4. click create 5. fill in datas 6. click save
TCACA0001	create a bank account	1. click module Accounting 2. select accounts 3. select Setup Your Bank Account 4. click create 5.choose Bank Type Account 6.fill in account number 7. click save
TCACT0001	create Texas	1. left click module Accounting 2. select Texas under Configurations 3. left click create 4. fill in fields 5. save texas
TCTSS0001	create a survey	1. left click module Tools 2. select Surveys 3. left click create 4. fill in datas 5. left click save
TCTSSR0001	create a survey request	1. left click module Tools 2. select Survey Requests 3. left click create 4. fill in datas 5. left click save
TCTSSP0001	create survey page	1. left click module Tools 2. select Survey Pages 3. left click create 4. fill in datas 5. left click save
TCPUPQ0001	creating a quotation	1. left click module Purchases 2. left click Quotations 3. left click create 4. fill fields 5. left click safe
TCPUPQ0002	delete all quotations	1. left click module Purchases 2. left click Quotations 3. mark all Quotations 4. left click more 5. left click delete

ID	Description	Steps
TCPUPPO0001	embed a purchase order	1. left click module Purchase 2. select Purchase Orders 3. mark a purchase order 4. left click more 5. left click embed 6. choose sharing message :Email 7. left click save
TCPUIPIS0001	receive incomming shipments	1. left click module Purchase 2. select incomming Shipments 3. open a incomming shipment 4. click receive
TCPUICODI0001	confirm draft invoices	1. left click module Purchase 2. select on draft invoices 3. mark a draft invoice 4. click more 5. click confirm draft invoice
TCPUICOIS0001	return an incoming shipment	1. left click module Purchase 2. select Incoming Shipment 3. mark the shipments, you want to return 4. click more 5. click return Shipments 6 fill in datas 7 click return
TCPUPP0001	create a product	1. left click module Purchase 2. select products 3. left click create 4. fill in fields 5. left click safe
TCHRHRE0001	create a employee	1. left click module Human Resources 2. select Employees 3. left click create 4. fill in datas 5. left click save
TCHRHRE0002	delete employees	1. left click module Human Resources 2. select Employees 3. mark some employees 4. click more 5. click delete
TCHREE0001	create expenses	1. left click module Human Resources 2. select Expenses 3. left click create 4. fill in Datas 5. left click save
TCHRLAR0001	approve a allocation request	1. left click module Human Resources 2. select Allocation Requests 3. choose a request 4. left click approve

ID	Description	Steps
TCHRAA0001	create a appraisal	1. left click module Human Resources 2. select Appraisal 3. click create 4. choose fields 5. click save
TCHRCJP0001	create a Job Position	1. left click module Human Resources 2. select Job Positions 3. left click create 4. fill in datas 5. left click save
TCHRCD0001	create a department	1. left click module Human Resources 2. select Department 3. left click create 4. fill in datas 5. left click save
TCPOSDOYS0001	open Your Session	1. left click module Point of Sale 2. left click Your Session
TCPOSDOYS0002	creating new session	1. left click module Point of Sale 2. left click Your Session 3. left click New Session
TCPOSDOYS0003	close session	1. left click module Point of Sale 2. left click Your Session 3. left click Close Session 4. left click Validate closing & post entries
TCPOSDOAS0001	put money in a session	1. left click module Point of Sale 2. select All Sessions 3. mark the sessions where you want to put money in 4. click more 5. click Put Money in
TCPOSPPC0001	open Product Catogories	1. left click module Point of Sale 2. left click Product Catogories
TCPOSPP0001	export Products	1. left click module Point of Sale 2. left click Products 3. mark products 4. click more 5. click export 6. click export to file
TCPOSCPOS0001	create a point of sale	1. left click module Point of Sale 2. left click Point of Sales 3. left click create 4. fill in datas 5. click save
TCPOSCPOS0002	set the created point of sale from TCPOS-0007 to inactive	1. left click module Point of Sale 2. left click Point of Sales 3. select a point of sale 4. click Set to Inactive

ID	Description	Steps
TCPOSCPM0001	embed Payment method	1. left click module Point of Sale 2. left click Payment Methods 3. mark payment method 4. click more 5. click embed 6. sharing method direct link or em-bed code 5.click share
TCPRPP0001	open Projects	1. left click module:Project 2. left click Projects
TCPRPP0002	create a new project	1. left click module:Project 2. left click Projects 3. fill in project name :test 4. left click save
TCPRPT0001	select Tasks	1. left click module:Project 2. left click Tasks
TCPRPT0002	create a new task	1. left click module:Project 2. left click Tasks 3.left click create 4. fill in taskname :taskname 5. left click save
TCPRPI0001	select Issues	1. left click module Project 2. left click Issues
TCPRPI0002	create a issue	1. left click module Project 2. left click Issues 3. left click create 4. fill in Issuename Issue 5. left click save
TCPRPI0003	mark issue as done	1. left click module Project 2. left click Issues 3. select the Issue :Issue 4. left click done
TCPRPI0004	cancel issue	1. left click module Project 2. left click Issues 3. select the Issue :Issue 4. left click Cancel Issue
TCPRLTPPP0001	Create project phase	1. left click module Project 2. left click Project Phases 3. left click create 4. fill in datas 5. left click save & close
TCPRLTPTP0001	Create team planing	1. left click module Project 2. left click Team Planning 3. left click create 4. fill in datas 5. left click save & close
TCPRSSP0001	Schedule Phases	1. left click module Project 2. left click Schedule Phases 3. fill in datas 4. left click compute

ID	Description	Steps
TCPRSST0001	Schedule Tasks	1. left click module Project 2. left click Schedule Tasks 3. fill in datas 4. left click compute
TCPRICTR0001	create a contract to renew	1. left click module Project 2. left click Contracts to Renew 3. left click create 4. fill in datas 5. left click save
TCPRIIT0001	create a invoice task	1. left click module Project 2. left click Invoice Task 3. left click create 4. fill in datas 5. left click save
TCPRCR0001	Create resource	1. left click module Project 2. unfold Resources 3. left click Resources 4. left click create 5. fill in datas 6. left click save
TCPRCRL0001	Create resource leave	1. left click module Project 2. unfold Resources 3. left click Resource Leaves 4. left click create 5. fill in datas 6. left click save
TCPRCWT0001	Create working time	1. left click module Project 2. unfold Resources 3. left click Working Time 4. left click create 5. fill in datas 6. left click save
TCFVV0001	create a vehicle	1. left click module Fleet 2. select Vehicles 3. Left click Create 4. fill in datas 5. left click save
TCFVVCON0001	delete vehicle contracts	1. left click module Fleet 2. select Vehicle Contracts 3. mark vehicles 4. left click more 5. left click delete
TCFVVCOS0001	set vehicle costs	1. left click module Fleet 2. select Vehicle Costs 3. Left click Create 4. set costs 5. left click save

Table A.1.: Successfully Executed Test Cases

Appendix B
Mixed-Tenancy: Changes over Time

B.1. Problem Analysis **191**
B.2. Computation of a Migration Plan **194**
B.2.1. Abstraction on a Problem of Graphs Theory 195
B.2.2. Complexity of Finding a Migration Plan 196
B.2.3. Computation of a Migration Plan 197
B.3. Conclusion **199**

Any change, even a change for the better, is always accompanied by drawbacks and discomforts.

Arnold Bennett

So far, this work has only dealt with the computation and realization of an initial deployment. In reality, however, it is very likely that a deployment that has been created at some point will need to change. This need to change may be caused, for example due to a changing customer base. This appendix presents some initial concepts for tackling this challenge.

This appendix is structured as follows. Section B.1 starts this appendix by analyzing the problem that shall be addressed. Based on this, Section B.2 discusses how the problem at hand may be tackled by abstracting it to a problem of graph theory. However, at this point it will be become evident that problem is in fact very complex and may not be solved easily. Thus, this section will also introduce an approach that allows to approximate an optimal solution for the problem. Section B.3 will close this appendix by drawing a conclusion.

The results, that are presented by this appendix, have been created in cooperation with Candide Orou-Yorouba. Candide created the results of this appendix in his Master's Thesis [OY13]. I supervised his thesis and guided him throughout the project.

B.1 Problem Analysis

So far in this work it has only been discussed how a Customer may express their Deployment Constraints, how a Valid and Optimal Deployment may be computed, and how this Deployment may initially be created. However, once created it is certain that the Deployment will have to change at some point. The need for changing the provisioned Deployment may be caused by the following triggers.

Changes of Customer base At some point it is very likely that the Customer base changes. This may either be since new Tenants want to use the application or existing Tenants choose not to use the application anymore.

Changes of Existing Deployment Constraints At some point it may be possible that a Tenants chooses to change their Deployment Constraints. Reasons for this may be manifold.

Customers Grouping Changes It may also be possible that the assignment of Tenant to Groups changes. This, for example, may be necessary since a Customer extends their area of business into a new industry or starts operating in a new geographic region.

The three reasons that have just been introduced may all require that a new Deployment is computed and realized.

Please note that it is not necessary to change the Deployment if new Users are introduced to the system or old ones are removed. This is due to the fact that it was one assumption of this work that it is possible to scale the infrastructure up and down according to the resource demand of Tenants.

However, once the Deployment Constraints have changed, the first step is to actually compute a new Valid and Optimal Deployment that applies to the new Deployment Constraints. This new deployment is captured by a new Deployment Configuration. For this work, the two deployments are referred to as Initial Deployment and Target Deployment.

Definition 31 (Initial Deployment) The deployment that has been realized and is running at the point of time when Deployment Constraints change, is referred to as *Initial Deployment*. It is the Deployment that shall be transformed into the Target Deployment.

Definition 32 (Target Deployment) The *Target Deployment* is the new Deployment that shall be created based on the new Deployment Information.

Once the Target Deployment has been computed, the next step is to actually introduce it.

When taking into consideration that the running application shall not be stopped, there are two strategies to do that. The first one realizes the Target on new infrastructure and migrate the Tenants from the old Deployment to the new one. The disadvantage of this strategy is that it will require the Operator to utilize lots of resources since at some point in time there will be two complete Deployments running. The second and favorable strategy is to sequentially transform the Initial Deployment into the Target Deployment through a series of Migration Operations. The advantage of this is that it is not necessary to create an entire new Deployment of the application.

Definition 33 (Migration Operations) A Migration Operation is a single, atomic activity that transforms a Deployment. Each Migration Operation has cost that is specific to the Operation, the Unit it is applied to, and the environment in which it is executed[1].

When conducting the research, it was possible to gain the conclusion that it is only necessary to distinguish the following six Migration Operations.

Creation of Unit A new Unit is created of a specific Deployment Level. This shall only be done if there exists no other Unit of this Deployment Level that has just been created and is not in use yet.

Deletion of Unit A Unit of a specific Deployment Level is withdrawn or deleted. This shall only be possible if there exist no higher Level Units that are hosted by the Unit to be deleted.

Movement of Unit A Unit of a specific Deployment Level is moved from one lower Level Unit to another. If this is done, all Units that are deployed on the moving Unit are moved as well.

Creation of new Tenant A new Tenant is registered at the application and the Units this Tenant is supposed to use.

Deletion of existing Tenant An existing Tenant is removed from the System. All its data is deleted or archived.

Movement of existing Tenant A Tenant's data is moved from one Unit to another. This may, obviously, only be done between Units of the same Deployment Level (and Application Components in case of Deployment Level dl_1).

For the realization of these Operations there are known patterns in literature. Some of them were defined and discussed as part of this research in [Feh+13]. However, the more interesting challenge within Mixed-Tenancy is to determine the necessary operations and their order to transform the Initial Deployment into the Target one. This is in fact the problem of creating the Migration Plan.

Definition 34 (Migration Plan) A Migration Plan is a sorted list of Migration Operations that migrate an Initial Deployment into the Target Deployment in an optimal way.

For the Migration Plan there are two things that shall be considered.

Target Constraints At any point of time the deployment of the application shall apply to the Deployment Constraints of the Target Deployment.

[1]Since the cost is specific to the environment, for this work it has been assumed that the cost is given from a cost function that needs to be implemented specifically for an environment.

Minimal Migration Cost In order to contribute to the Operator's interest, the optimization criteria for the algorithm computing the Migration Plan shall be that a Migration Plan only causes minimal cost. The cost of a Migration Plan is the sum of costs of all Migration Operations.

Figure B.1 gives a trivial example. In the Initial Deployment of the example there are two Tenants each using their own Unit of an Application Component. In the Target Deployment both Tenants share the same Unit. To perform this

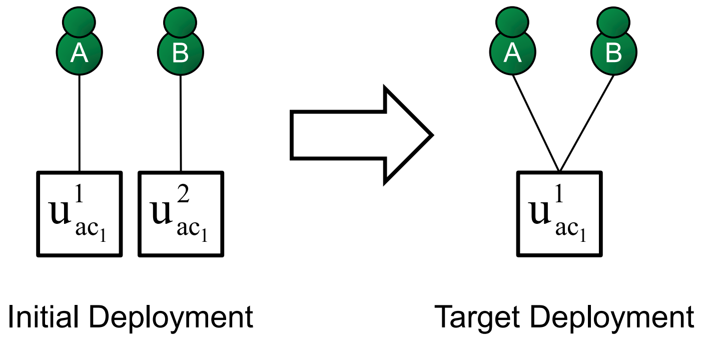

Figure B.1.: Trivial Example Migration

transformation, there are three Migration Plans possible.

1. Migrate Tenant A to Unit $u^2_{ac_1}$.
2. Migrate Tenant B to Unit $u^1_{ac_1}$.
3. Create a new Unit $u^3_{ac_1}$.
   Migrate both Tenants to this Unit $u^3_{ac_1}$.

All three Plans would be possible Migration Plans to perform the transformation. When thinking about which one would be the most optimal, it is quite obvious that Migration Plan 3 will always require more cost than 1 and 2. However, whether Migration Plan 1 or 2 is the less costly one, may depend on many factors (e.g. data volume attached to Tenants).

Thus, the question to be addressed by this appendix is how an optimal Migration Plan can be found for a given scenario.

B.2 Computation of a Migration Plan

Based on the problem analysis of the previous section, this section will present a solution approach. The idea of this approach is to abstract the introduced problem of finding a Migration Plan to a graph. However, if this is done, a problem of significant complexity arises.

The section starts with the discussion how the problem of finding an optimal Migration Plan may be abstracted to a graph (Subsection B.2.1). Based on this, the complexity of solving the problem is analyzed by Subsection B.2.2. Concluding this section, Subsection B.2.3 introduces an algorithm that strives to find an optimal Migration Plan.

B.2.1 Abstraction on a Problem of Graphs Theory

In the problem analysis a trivial migration example was given (Figure B.1). For this example, a total of three possible Migration Plans would be possible and it was stated that which one is the optimal Migration Plan depends on the cost of the Migration Operations. It is quite obvious that the number of possible Migrations Plans will rise with a more complex example. Furthermore, the trivial example only consisted of a single Deployment Level. However, as it was discussed in this work, there may be multiple Deployment Levels.

In order to investigate this further, Figure B.2 illustrates a more complex example. In this example there is a total of five Tenants. The Tenants A, B, and

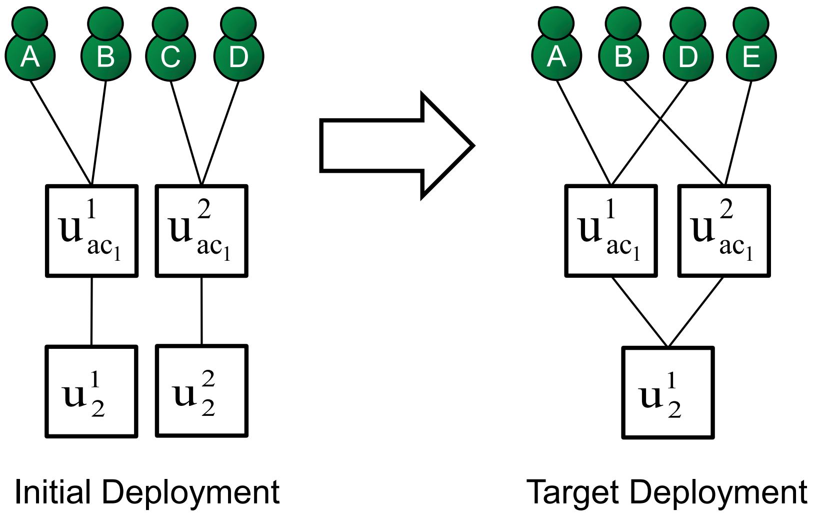

Figure B.2.: A more complex Example Migration

D are in both, the Initial and the Target Deployment. Tenant C is only in the Initial Deployment. Thus, it appears like they have canceled their subscription and will stop using the application. Tenant E, on the other hand, starts using the application with the Target Deployment as they are not part of the Initial Deployment.

A Migration Plan may be created by executing the following steps.

Deletion of existing Tenants The first step that can be taken is to delete those existing Tenants that are not contained in the Target Deployment. It is reasonable to do that as a first step since once deleted, these Tenants do not need to be considered anymore for further steps.

Transform of Deployment Once the Tenants that have unsubscribed, it is necessary to transform the Initial Deployment into the Target one. This is done by not only considering those Tenants that are part of both the Initial Deployment and the Target Deployment. This step will be analyzed in more detail shortly.

Creation of new Tenants The last step should be to create the new Tenants (Tenant E in the example). This should be done last since otherwise it would be necessary to deal with their Deployment Constraints when rearranging the Deployment.

Based on this the open question is how can an optimal Migration Plan be found for the Tenants that are using both, the Initial as well as the Target Deployment. The idea to do that is to create a graph that contains all possible Migration Plans. This graph is called Migration Graph.

Definition 35 (Migration Graph) A *Migration Graph* is a directed weighted multigraph in which Deployments are represented as vertices and edges represent Migration Operations that transform a Deployment into its neighbor. Each edge has the cost of the Operation attached as weight.

Of course, the Migration Graph only contains those Migration Plans that apply to the Deployment Constraints of the Target Deployment[2]. This contributes to the introduced condition that while the running Deployment is changed, it shall never go through a state that does not apply to the Deployment Constraints of the Target Deployment.

Once a Migration Graph has been created, it is possible to determine the optimal Migration Plan by determining the cheapest path between the Initial Deployment and the Target Deployment. Finding the cheapest path in a graph is a well-known problem in graph theory and there are many algorithms available (e.g. Dijkstra, A* [RN10]). In Figure B.3 the cheapest path between the Initial and the Target Deployment introduced assuming that the cost of all Migration Operations are equal[3]. This path is the optimal transformation of the deployment and is part of the Migration Plan. It consists of the following steps if it is assumed that all Migration Operations cause equal cost.

An algorithm that is capable of creating the entire graph with considering the Deployment Information of the Target Deployment may be found in [OY13]. Unfortunately, this approach will only work for small scenarios. This will be further elaborated in the next subsection.

B.2.2 Complexity of Finding a Migration Plan

In the previous section an approach has been presented to compute an optimal Migration Plan by creating a Migration Graph that includes all possible Deployments. Unfortunately, this approach will only work for small examples since the number of vertices increases very fast with a rising number of Tenants. In [OY13] a mathematical model was presented that allows to calculate an upper bound for the number of vertices based on a number of Tenants and Deployment Levels. Figure B.4 illustrates how many vertices are generated based on a number of Tenants in a setting that has only two Deployment Levels. As visible in

[2]Please note that this is not considered in the example, as it would require a more detailed discussion. The goal of this appendix is to only give an overview. A full discussion of it can be found in [OY13]

[3]Please note that in the final step illustrated by Figure B.3 there is also a renaming of the Units done. This step is actually not necessary since it does not change the structure of the deployment. Thus, it is not specifically marked.

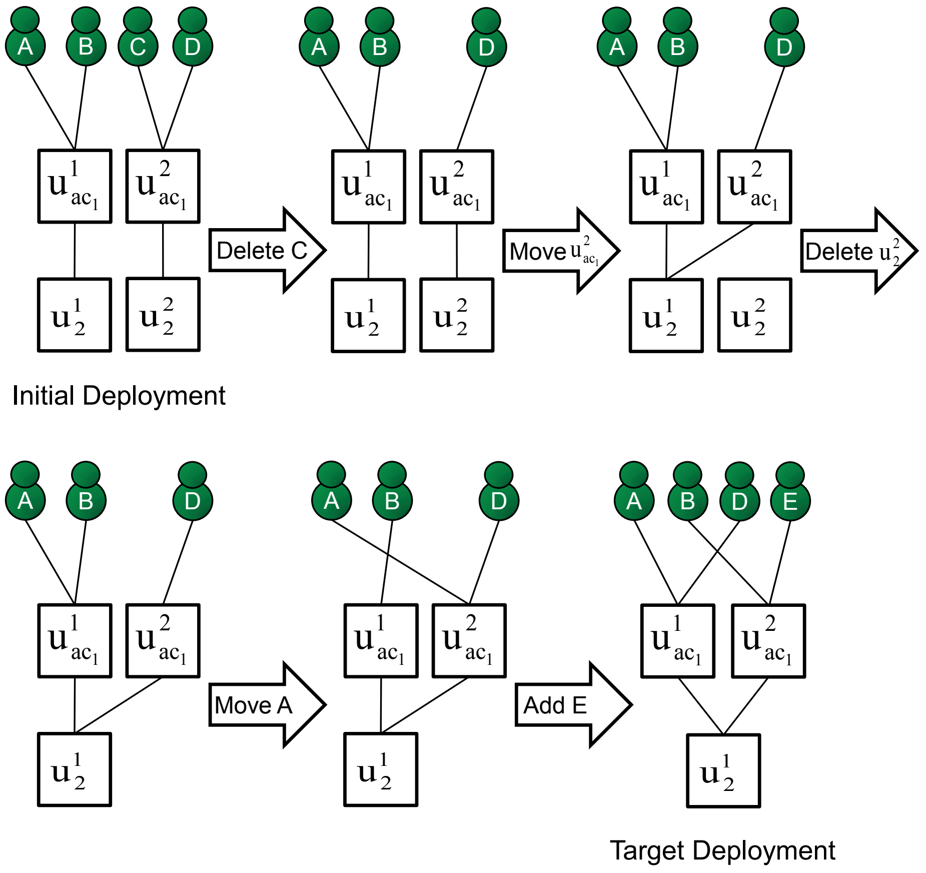

Figure B.3.: Optimal Migration Plan for more Complex Example

the figure the number of Tenants increases exponentially. To be more precise a deployment with ten Tenants will generate a Migration Graph that contains about 68,000,000 Vertices. Keeping such a huge graph in memory will not work especially considering the growth rate. Even if it was possible to come up with enough memory, the problem would still not be solvable by this approach since the run-time increases proportionally to the growth of the graph as well. Thus, it will not be possible to address this issue by using more hardware.

In fact, it is necessary to come up with an approach that does not require the computation of the entire Migration Graph. Such an approach is briefly presented in the next subsection.

B.2.3 Computation of a Migration Plan

In the previous subsection, it has been analyzed that the problem of computing a Migration Plan may be solved by abstracting the problem to a problem of graph theory. In fact, it was possible to show that the problem may be solved by computing the cheapest path in a directed multigraph. Unfortunately, however, it was also possible to show that the size of the graph grows exponentially, and, thus, it is not possible to perform an extensive/brute force search.

In literature, online algorithms are algorithms that allow to solve a given prob-

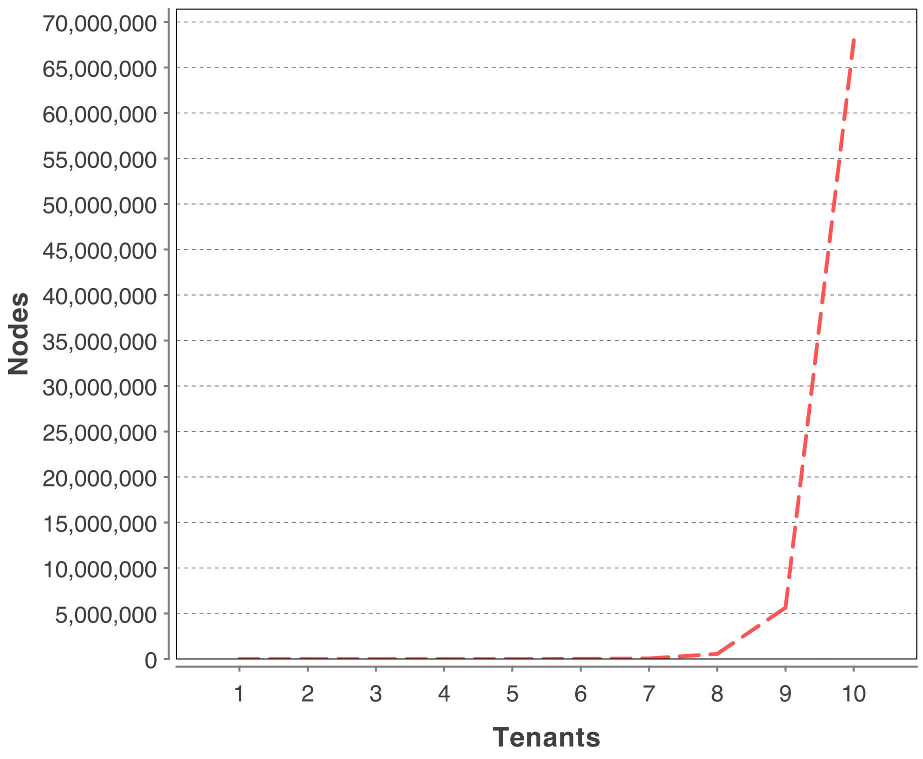

Figure B.4.: Vertices vs. Tenants (two Deployment Levels) [OY13]

lem without having the entire necessary input available from the start [BEY98]. This may, for example, be due to the fact that the entire input is far to huge in order too be processed at once. Thus, in [OY13], Candide analyzed several available online algorithms to tackle the problem of finding a Migration Plan. He identified the *Real-Time A** algorithm ([Kor90]) as the most suitable of the investigated algorithms.

Real-Time A* is a heuristic that strives to find a cheapest path on an unknown map. Therefore, it makes use of a merit function $f(n) = g(n) + h(n)$ to evaluate a given node n. For this function $h(n)$ represents the heuristic function which indicates the estimated cost of reaching the target node from the current node n. $g(n)$ represents the actual cost of reaching any given node n from the current node, rather than from the root node as it is the case with the well-known offline A* Algorithm.

Using the merit function, Real-Time A* determines the $f(n)$ of all neighboring nodes and moves to the neighbor with the best results. This means that the new n is set to be the neighbor with the best $f(n)$. In fact, that is not just done for the direct neighboring notes but for all neighbors within a certain lookahead depth of the current node (e.g. in case of lookahead depth 2, all nodes are reachable by two moves – along the edges). These steps are repeated until the Target Deployment is found. This was only a very primitive description of the approach, a detailed one may be found in [OY13].

In Figure B.5 the execution time of Real-Time A* is compared to the naive brute force approach of creating the entire graph and applying the A* algorithm. This is done based on examples with two to ten Tenants. As visible in the figure at

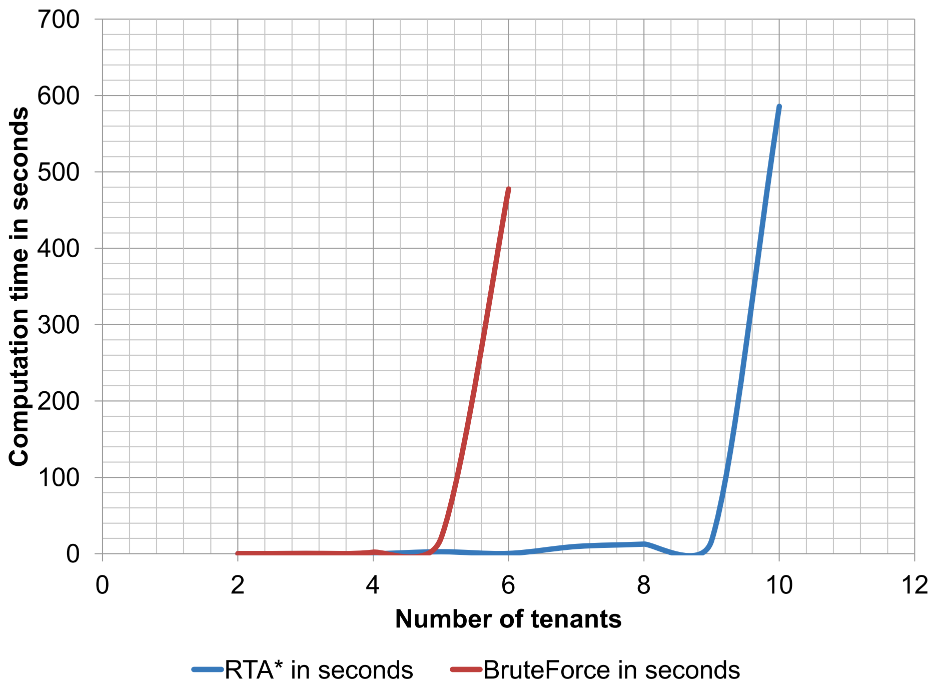

Figure B.5.: Real-Time A* (Lookahead depth = 1) vs. Brute-Force [OY13]

a lower number of Tenants (up to five), computation time is quite the same. However, already for six Tenants the Real-Time A* algorithm performs much faster. Thus, the Real-Time A* approach allows to solve bigger problems than the brute force approach. However, even for the Real-Time A* execution time increases significantly at a given point.

With respect to quality the Real-Time A* and the brute force algorithm have been compared based on a single problem instance (introduced in [OY13]). For this example, a lookahead depth of one to be used for the Real-Time A*. This revealed that in the context of the example, the average competitive ratio of 1.2 was achieved by Real-Time A*. This means that in average the Migration Plan created using Real-Time A* causes 20% more cost than the brute force Migration Plan for the given problem instance.

B.3 Conclusion

This appendix summarized the results that were created in cooperation with Candide Ourou-Yorouba. In his Master's Thesis ([OY13]) Candide addressed the issue that Deployments have to change due to changing Deployment Constraints. This was focused on the creation of a Migration Plan describing how a Deployment that has been realized and is running (Initial Deployment) may be transformed into a Deployment that applies to the new Constraints (Target Deployment). This transformation shall be executed in the way that there is no point in time where the running Deployment does not apply to the Deployment Constraints of the Target Deployment. Furthermore, it is the optimization criteria that the transformation shall be done by causing minimal cost for the Operator.

As discussed, it is possible to abstract the problem of finding an optimal Migration Plan to the problem of finding a cheapest path in the Migration Graph. The Migration Graph is a directed weighted multigraph that represents Deployment as vertices and Migration Operations (e.g. creation of a Unit, movement of a Unit) as edges.

However, the problem with this approach is that the number of vertices in the Migration Graph raises exponentially. Thus, it is not possible to create the entire Migration Graph in memory and perform a cheapest path search on it. A review of available online algorithms was conducted. It revealed that from the investigated approaches, Real-Time A^* is the most suitable. A comparison to the brute force approach revealed that for one given investigated problem instance, Real-Time A^* produces an average competitive ratio of 1.2. With respect to execution time, it was analyzed that it is quite the same for problems with up to four Tenants. After that, the Real-Time A^* is significantly faster. However, so far the biggest problems instances investigated contained a total of ten Tenants. Creating algorithms for tackling problems with more Tenants is up to future research. Furthermore, so far evaluation has purely been based on a single problem instance. It is up to future research to investigate the problem of Migration Plan creation more thoroughly.

In order to be able to alter the application's deployment according to the Migration Plan it would be necessary to change the Deployment Platform. Since the Mixed-Tenancy Deployment Platform proposed by this work utilizes Chef, the necessary changes would be minimal. However, if this is actually possible is still open for further investigation.

Another shortcoming of the approach presented in this appendix is that it takes the Target Deployment as input. An alternative for this would be that the Target Deployment is actually determined by the algorithm computing the Migration Plan. This would allow to search for a Valid and Optimal Target Deployment that can be reached with minimal cost. However, since computing a Valid and Optimal Deployment is already a very complex problem, this has not been considered by this appendix and is still up for future research.

Appendix C
Optimization considering limited Resource Availability

C.1. Problem Analysis 201
C.2. Introduction of Bin-Packaging (with Conflicts) 204
C.3. Conclusion 205

Simplicity is a great virtue but it requires hard work
to achieve it and education to appreciate it.
And to make matters worse: complexity sells better.

Edsger Wybe Dijkstra

Chapter 5 analyzed the challenge of computing a Valid and Optimal Deployment based on the Deployment Information. In Section 5.2.1, which analyzed the Mixed-Tenancy Deployment Problem, it was stated that for this work it is assumed that any Unit may host any number of higher Level Units without having a restriction to resource availability. It is the purpose of this appendix to give some insight on how not making this assumption would alter the optimization problem. It is not the purpose of this appendix to discuss the problems in detail but only to give some insight.

In order to do that the appendix starts with a problem analysis presented by C.1. Based on this analysis, Section C.2 discussed the challenges involved with computing an optimal solution if resource limitations shall be considered. This appendix is closed by Section C.3 with a description of a conclusion.

C.1 Problem Analysis

When the problem of computing a Valid and Optimal Deployment was analyzed in Section 5.2.1, it was stated that it is assumed for this work that there are no resource limitations. This means that any Unit may host an infinite number of higher Level Units. It is still the case that a Valid and Optimal Deployment is optimal if it utilizes minimal resources.

Even though the definition of optimal does not need to change, it is necessary to redefine what a Valid Deployment is. The only characteristic a Valid Deployment has had to realize so far was that it has to apply to all Deployment

Constraints. However, if resource limitations shall be considered there is a second characteristic that determines if a Deployment is a Valid Deployment.

Resource Constraints are Considered Every piece of infrastructure (or Unit of a Deployment Level) only provides a limited amount of resources. Thus, the demand for hosted resources of upper Deployment Level Units shall not be higher than the amount of resources that can be provided.

In order to be able to verify if this characteristic is actually met, it is necessary to define new inputs for the new Mixed-Tenancy deployment problem. These are described by the following. In this description the number of resources or the resource demand is always represented by a positive real number.

Resource Limitation This input defines the maximum number of resources a Unit may host. Since this may be different for every type of Units, it is necessary that it is being defined for every Application Component and all Deployment Levels (except Level dl_1).

Resource Demand of Tenant Each Tenant has a certain resource demand for using every Application Component. It has previously been defined in Section 5.2.1. This demand depends, for example, on the number of Users of this Tenant. Since it is specific to a Tenant, it must be expressed per Tenant for every Application Component.

Resource Demand of Unit Similar to the demand of Tenants, also every Unit has a resource demand. It has previously been defined in Section 5.2.1. This demand refers to the overhead cost. Since it is specific to each type of Unit, it is necessary to define it for every Application Component and all Deployment Levels (except Level dl_1).

In order to determine the number of resources a Unit is hosting, it is necessary to sum up the resource demand all hosted Units and add the resource demand of the Tenants that are using the Level dl_1 Units.

Furthermore, there is one more thing that needs to be mentioned. It is assumed at this point that a single Tenant's resource demand of may always be met by a single Unit. This leads to the conclusion that the Resource Limitation of every type of Unit is at least as big as the Resource Demand of the Tenant with the highest Resource Demand.

In order to elaborate on the structure of a deployment and its resource demand further Figure C.1 and the Tables C.1, C.2, and C.3 illustrate an example. In Table C.1 it is stated for example that a Unit of Application Component ac_1 may only host up to 10 resources and a Unit of Level dl_2 may only host a total of 11 resources. Table C.2 states the resource demand of all Tenants for every Application Component. For example, the Tenants A and B both require five resources of Application Component ac_1, for Application Component ac_2 all Tenants require one resource. In Table C.3 the Resource Demands for every type of Unit is stated. In order to have a simple and intuitive example, the Resource Demand of all types of Units is assumed to be one. These were the inputs that were required for the

	dl_1	dl_2
ac_1	10	11
ac_2	5	

Table C.1.: Example Resource Limitation

	A	B	C	D
ac_1	5	5	2	2
ac_2	1	1	1	1

Table C.2.: Example Resource Demand of Tenant

new characteristic of a Valid Deployment. However, it is also necessary to have the Deployment Information as an input. It is visualized on the left hand side of Figure C.1. One possible Valid and Optimal Deployment that was created based

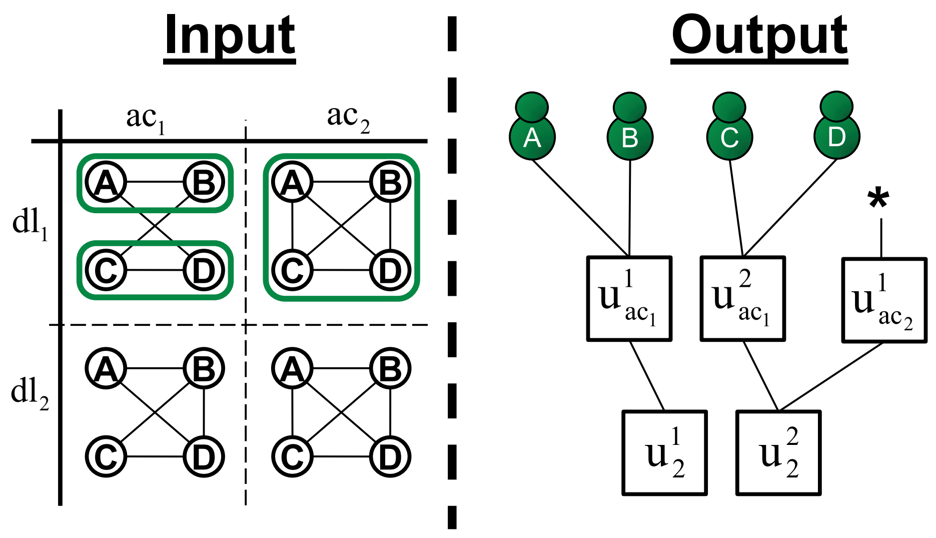

Figure C.1.: An Example for a Valid and Optimal Deployment

on all these inputs, is illustrated on the right hand side of Figure C.1. As visible it applies to the Deployment Information. Furthermore, it also applies to all resources' constraints. As an example, take Unit u_1^2. According to Table C.1, a Unit of this type may host a total of 11 resources. It does host one Unit of Application Component ac_1 (resource demand: 1), one Unit of Application Component ac_2 (resource demand: 1), Tenant C and D for Application Component ac_2 (resource demand: 4) and all Tenants for Application Component ac_2 (resource demand: 4). If the individual resource demands are summed up, the total resource demand hosted on the Unit is 10. Since the Unit may host 11 resources, the Deployment is Valid at this point according to the resource characteristic.

Besides being Valid, the Deployment is also Optimal. This has manually been checked.

	dl_1	dl_2
ac_1	1	1
ac_2	1	

Table C.3.: Example Resource Demand of Unit

C.2 Introduction of Bin-Packaging (with Conflicts)

In Chapter 5 it was possible to show that the problem of assigning Tenants to Units is in fact the well-known NP-hard graph theory problem called clique cover (Lemma 1). However, clique cover can only solve the problem of finding a minimal number of Units with respect to the Deployment Information. It does not cover any constraints with respect to resource consumption.

In literature there exists another well-known NP-hard problem called bin-packaging [AM98; CJGJ96]. In the bin-packing problem, there is a set of objects each haveing a certain volume. These objects are to be packed into finite number of bins or containers that each may only hold a certain volume of objects. The optimization problem of bin-packaging is to minimize the number of bins.

Bin-packaging allows to assign Tenants to a minimal number of Units by only considering to the resource limitation constraint. The only thing to do is to think of Units as bins. Each bin must have a maximum volume it may hold. So does a Unit – it only offers a certain amount of resources. The objects in bin-packaging correspond to the Tenant. Where the objects have a certain volume, Tenants have a certain Resource Demand. Thus, it is possible to say that the problem of assigning Tenants to a minimal number of Units by applying to the resource limitation constraint is, in fact, the bin-packaging problem.

Unfortunately, this only solves half of the problem as it only allows to apply to the resource limitation constraint. The Deployment Information has not been considered yet. In the thinking of bin-packaging the Deployment Constraints would correspond to the idea that there are constraints that permit or prohibit certain objects to share a bin. Bin-packaging with these special constraints is, in fact, also known in literature as a special version of bin-packaging called bin-packaging with conflicts [JÖ97; Jan99]. Conflicts refer to the constraints that some objects may not be assigned to the same bin. In the context of this appendix, this translates to the constraint that some combination of Tenants may not be ascending to the same Unit.

All this leads to the conclusion that the problem of assigning Tenants to Units by considering both, the resource limitation constraint and the Deployment Information, is in fact the same problem as bin-packaging with conflicts. Since bin-packaging with conflicts is a special more complicated version of the normal bin-packaging problem, it is also NP-hard [JÖ97]. There are known approximation algorithms known in literature (e.g. [Gup+08; Fer+11; SVD03; SV13]). However, due to the fact that the problem is much less known, there are no implementations of them available in standard graph frameworks.

C.3 Conclusion

In Chapter 5 the problem of computing a Valid and Optimal Deployment based on the Deployment Information was analyzed. In this chapter it was assumed that there are no resource restrictions to consider and, thus, it is possible that any Unit may host an infinite number of higher Level Units. In this appendix it was presented that by not having this assumption will increase the complexity of the optimization problem.

The first step towards doing this was to analyze the problem. Based on this problem analysis, it was possible to show that the problem of assigning Tenants by considering both, the Deployment Information and resource restrictions, is in fact the problem called bin-packaging with conflicts. Bin-packaging with conflicts is known in literature and is proven to be NP-hard.

However, the impact of the identification of the bin-packaging with conflicts problem is open for future research.

Bibliography

[ADG02] R. Anzböck, S. Dustdar, and H. Gall. "Software configuration, distribution, and deployment of web-services." In: *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. SEKE '02. ACM ID: 568872. New York, NY, USA: ACM, 2002, 649–656. ISBN: 1-58113-556-4. DOI: `10.1145/568760.568872` (cited on page 34).

[AGI12] M. Almorsy, J. Grundy, and A. Ibrahim. "TOSSMA: A Tenant-Oriented SaaS Security Management Architecture." In: *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. 2012, pp. 981–988. DOI: `10.1109/CLOUD.2012.146` (cited on page 32).

[AH11] D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in Rdfs and Owl*. 2nd revised edition. Morgan Kaufmann, July 2011. ISBN: 0123859654 (cited on pages 69, 72).

[AH77] K. Appel and W. Haken. "Every planar map is four colorable. Part I: Discharging." In: *Illinois Journal of Mathematics* 21.3 (1977), 429–490 (cited on page 91).

[AK12] M. R. Aswin and M. Kavitha. "Cloud intelligent track - Risk analysis and privacy data management in the cloud computing." In: *2012 International Conference on Recent Trends In Information Technology (ICRTIT)*. April 2012, pp. 222–227. DOI: `10.1109/ICRTIT.2012.6206752` (cited on page 39).

[AM98] S. Albers and M. Mitzenmacher. "Average-case analyses of first fit and random fit bin packing." In: *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*. 1998, 290–299 (cited on page 204).

[Arm+09] M. Armbrust, A. Fox, R. Griffith, A. D Joseph, R. H Katz, A. Konwinski, G. Lee, D. A Patterson, A. Rabkin, I. Stoica, et al. "Above the clouds: A berkeley view of cloud computing." In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28* (2009) (cited on page 23).

[Aul+08] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. "Multi-tenant databases for software as a service: schema-mapping techniques." In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. SIGMOD '08. New York, NY, USA: ACM, 2008, 1195–1206. ISBN: 978-1-60558-102-6 (cited on page 32).

[BAT12] W. Brown, V. Anderson, and Q. Tan. "Multitenancy - Security Risks and Countermeasures." In: *2012 15th International Conference on Network-Based Information Systems (NBiS)*. 2012, pp. 7–13. DOI: `10.1109/NBiS.2012.142` (cited on page 32).

[BEY98] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. New York, NY, USA: Cambridge University Press, 1998. ISBN: 0-521-56392-5 (cited on page 198).

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. "Free Bits, PCPs, and Nonapproximability—Towards Tight Results." In: *SIAM Journal on Computing* 27.3 (1998), 804–915 (cited on page 93).

[BHK09] A. Björklund, T. Husfeldt, and M. Koivisto. "Set partitioning via inclusion-exclusion." In: *SIAM Journal on Computing* 39.2 (2009), 546–563 (cited on page 93).

[BR12] R Balakrishnan and K Ranganathan. *A Textbook of Graph Theory*. English. New York, NY: Springer New York : Imprint: Springer, 2012. ISBN: 97814614-45296 1461445299 9781461445289 1461445280 (cited on pages 44, 89, 91, 93).

[Bre79] D. Brelaz. "New methods to color the vertices of a graph." In: *Communications of the ACM* 22.4 (1979), 251–256 (cited on page 93).

[Bro41] R. L. Brooks. "On colouring the nodes of a network." In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 37. 1941, 194–197 (cited on page 93).

[Buy+09] R. Buyya, C. S Yeo, S. Venugopal, J. Broberg, and I. Brandic. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." In: *Future Generation Computer Systems* 25.6 (2009), 599–616 (cited on page 23).

[BZ10] C.-P. Bezemer and A. Zaidman. "Multi-tenant SaaS applications: maintenance dream or nightmare?" In: *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*. IWPSE-EVOL '10. New York, NY, USA: ACM, 2010, 88–92. ISBN: 978-1-4503-0128-2. DOI: `10.1145/1862372.1862393` (cited on pages 1, 13, 14).

[Cao+11] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou. "Privacy-Preserving Query over Encrypted Graph-Structured Data in Cloud Computing." In: *2011 31st International Conference on Distributed Computing Systems (ICDCS)*. June 2011, pp. 393–402. DOI: `10.1109/ICDCS.2011.84` (cited on page 38).

[Cao+14] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. "Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data." In: *IEEE Transactions on Parallel and Distributed Systems* 25.1 (January 2014), pp. 222–233. ISSN: 1045-9219. DOI: `10.1109/TPDS.2013.45` (cited on page 38).

[CC06] F. Chong and G. Carraro. "Architecture Strategies for Catching the Long Tail." In: *Microsoft MSDN* (April 2006). URL: `http://msdn.microsoft.com/en-us/library/aa479069.aspx` (visited on March 4, 2010) (cited on pages 1, 2, 12–14).

[CCW06] F. Chong, G. Carraro, and R. Wolter. "Multi-tenant data architecture." In: *MSDN Library, Microsoft Corporation*. 2006 (cited on pages 14, 30, 31).

[CD06] K. D. Cooper and A. Dasgupta. "Tailoring graph-coloring register allocation for runtime compilation." In: *Proceedings of the International Symposium on Code Generation and Optimization*. 2006, 39–49 (cited on page 92).

[Cha+11] D. Chadwick, S. Lievens, J. Den Hartog, A. Pashalidis, and J. Alhadeff. "My Private Cloud Overview: A Trust, Privacy and Security Infrastructure for the Cloud." In: *2011 IEEE International Conference on Cloud Computing (CLOUD)*. July 2011, pp. 752–753. DOI: `10.1109/CLOUD.2011.113` (cited on page 38).

[CJGJ96] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson. "Approximation algorithms for bin packing: A survey." In: *Approximation algorithms for NP-hard problems*. 1996, 46–93 (cited on page 204).

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. English. Cambridge, Masachusetts; London: The MIT Press, 2009. ISBN: 0262033844 (cited on page 90).

[CSL09] X. Cheng, Y. Shi, and Q. Li. "A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA." In: *2009 Joint Conferences on Pervasive Computing (JCPC)*. December 2009, pp. 599–604. DOI: `10.1109/JCPC.2009.5420114` (cited on page 33).

[DBV05] E. Dolstra, M. Bravenboer, and E. Visser. "Service configuration management." In: *12th International Workshop on Software Configuration Management (SCM-12)*. 2005 (cited on page 34).

[DM11] C. Dabrowsk and K. Mills. "VM Leakage and Orphan Control in Open-Source Clouds." In: *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*. November 2011, pp. 554–559. DOI: `10.1109/CloudCom.2011.84` (cited on page 38).

[Fay13] A. Fayolle. *Bug #1030795 "stock_picking.action_invoice_create: calling by x..." : Bugs : OpenERP Server*. November 2013. URL: `https://bugs.launchpad.net/openobject-server/+bug/1030795` (visited on November 9, 2013) (cited on page 157).

[Feh+13] C. Fehling, F. Leymann, S. T. Ruehl, M. Rudek, and S. Verclas. "Service Migration Patterns – Decision Support and Best Practices for the Migration of Existing Service-Based Applications to Cloud Environments." In: *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications (SOCA)*. December 2013, pp. 9–16. DOI: `10.1109/SOCA.2013.41` (cited on pages 99, 193).

[Feh+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. English. Springer, 2014. ISBN: 978-3-7091-1568-8 (cited on page 10).

[Feh09] C. Fehling. *Provisioning of software as a service applications in the cloud*. Tech. rep. University of Stuttgart, October 2009. URL: `http://elib.uni-stuttgart.de/opus/volltexte/2009/4766/` (visited on September 22, 2012) (cited on pages 21, 33).

[Fer+11] A. Fernández, C. Gil, A. L. Márquez, R. Baños, M. G. Montoya, and M. Parra. "A memetic algorithm for two-dimensional multi-objective bin-packing with constraints." In: *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*. GECCO '11. New York, NY, USA: ACM, 2011, 341–346. ISBN: 978-1-4503-0690-4. DOI: `10.1145/2001858.2002016` (cited on page 204).

[FLM10] C. Fehling, F. Leymann, and R. Mietzner. “A Framework for Optimized Distribution of Tenants in Cloud Applications.” In: *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*. July 2010, pp. 252 –259. DOI: `10.1109/CLOUD.2010.33` (cited on pages 21, 33, 167).

[GHJ94] E. Gamma, R. Helm, and R. E. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software.* 1st ed., Reprint. Addison-Wesley Longman, Amsterdam, October 1994. ISBN: 0201633612 (cited on pages 59, 63, 71, 140, 141, 169).

[GKS13] T. Garg, R. Kumar, and J. Singh. “A way to cloud computing basic to multitenant environment.” In: *International Journal of Advanced Research in Computer and Communication Engineering Vol. 2* Issue 6, June 2013 (June 2013) (cited on page 14).

[GRV04] R. L. Glass, V. Ramesh, and I. Vessey. “An analysis of research in computing disciplines.” In: *Communications of the ACM* 47.6 (2004), 89–94 (cited on pages 4, 5).

[GS06] J. Gebauer and F. Schober. “Information system flexibility and the cost efficiency of business processes.” In: *Journal of the Association for Information Systems* 7.3 (2006), 122–147 (cited on pages 33, 34).

[Guo+07] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. “A Framework for Native Multi-Tenancy Application Development and Management.” In: *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*. Tokyo, Japan, July 2007, pp. 551–558. DOI: `10.1109/CEC-EEE.2007.4` (cited on page 32).

[Gup+08] R. Gupta, S. Bose, S. Sundarrajan, M. Chebiyam, and A. Chakrabarti. “A Two Stage Heuristic Algorithm for Solving the Server Consolidation Problem with Item-Item and Bin-Item Incompatibility Constraints.” In: *IEEE International Conference on Services Computing, 2008. SCC ’08*. Vol. 2. 2008, pp. 39–46. DOI: `10.1109/SCC.2008.39` (cited on page 204).

[GVR02] R. L. Glass, I. Vessey, and V. Ramesh. “Research in software engineering: an analysis of the literature.” In: *Information and Software technology* 44.8 (2002), 491–506 (cited on pages 4, 5).

[Hal93] M. M. Halldórsson. “A still better performance guarantee for approximate graph coloring.” In: *Information Processing Letters* 45.1 (1993), 19–23 (cited on page 93).

[Hut11] M. Huth. *Logic in computer science: modelling and reasoning about systems*. English. Cambridge [u.a.: Cambridge Univ. Press, 2011. ISBN: 052154310X 9780521543101 (cited on pages 7, 44).

[Hüt12] M. Hüttermann. *DevOps for developers*. English. [New York]: Apress : Distributed to the book trade worldwide by Springer Science+Business Media New York, 2012. ISBN: 9781430245698 1430245697 (cited on page 153).

[Jan99] K. Jansen. “An Approximation Scheme for Bin Packing with Conflicts.” en. In: *Journal of Combinatorial Optimization* 3.4 (December 1999), pp. 363–377. ISSN: 1382-6905, 1573-2886. DOI: `10.1023/A:1009871302966` (cited on page 204).

[JM08] K. Jansen and M. Margraf. *Approximative Algorithmen und Nichtapproximierbarkeit*. German. Berlin; New York, NY: De Gruyter, 2008. ISBN: 9783110203165 3110203162 (cited on pages 19, 90, 91).

[JT95] T. R. Jensen and B. Toft. *Graph coloring problems*. Vol. 39. John Wiley & Sons, 1995 (cited on page 91).

[JÖ97] K. Jansen and S. Öhring. “Approximation algorithms for time constrained scheduling.” In: *Information and Computation* 132.2 (1997), 85–108 (cited on page 204).

[Kar72] R. M. Karp. *Reducibility among combinatorial problems*. Tech. rep. Springer, 1972. (Visited on July 17, 2013) (cited on pages 92, 94).

[Klo02] W. Klotz. “Graph coloring algorithms.” In: *Mathematics Report* (2002), 1–9 (cited on page 94).

[KM08] T. Kwok and A. Mohindra. “Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications.” In: *Service-Oriented Computing – ICSOC 2008*. Ed. by A. Bouguettaya, I. Krueger, and T. Margaria. Lecture Notes in Computer Science 5364. Springer Berlin Heidelberg, January 2008, pp. 633–648. ISBN: 978-3-540-89647-0 (cited on page 32).

[KMMG13] C. Kalloniatis, V. Manousakis, H. Mouratidis, and S. Gritzalis. “Migrating into the Cloud: Identifying the Major Security and Privacy Concerns.” In: *Collaborative, Trusted and Privacy-Aware e/m-Services*. Springer, 2013, 73–87 (cited on pages 1, 26).

[KN09] S. O. Krumke and H. Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. German. Wiesbaden: Vieweg+Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden, 2009. ISBN: 9783834895929 383489592X (cited on pages 89, 91).

[Kor90] R. E. Korf. “Real-time heuristic search.” In: *Artificial Intelligence* 42.2–3 (March 1990), pp. 189–211. ISSN: 0004-3702. DOI: `10.1016/0004-3702(90)90054-4` (cited on page 198).

[Koz11] H. Koziolek. “The SPOSAD Architectural Style for Multi-tenant Software Applications.” In: *2011 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. 2011, pp. 320–327. DOI: `10.1109/WICSA.2011.50` (cited on page 32).

[Kub04] M. Kubale. *Graph Colorings*. en. American Mathematical Soc., 2004. ISBN: 9780821856871 (cited on pages 89, 91–94).

[LDPS10] F. Lombardi, R. Di Pietro, and C. Soriente. “CReW: Cloud Resilience for Windows Guests through Monitored Virtualization.” In: *2010 29th IEEE Symposium on Reliable Distributed Systems*. October 2010, pp. 338–342. DOI: `10.1109/SRDS.2010.48` (cited on page 38).

[LHLP10] C. Lizhen, W. Haiyang, J. Lin, and H. Pu. “Customization modeling based on metagraph for multi-tenant applications.” In: *5th International Conference on Pervasive Computing and Applications*. Maribor, Slovenia, December 2010, pp. 255–260. DOI: `10.1109/ICPCA.2010.5704108` (cited on page 34).

[LMRV14] S. Lange, M. Margraf, S. T. Ruehl, and S. A. W. Verclas. "On Valid and Optimal Deployments for Mixed-Tenancy Problems in SaaS-Applications." In: *2014 IEEE 10th World Congress on Services (Services)*. July 2014 (cited on page 89).

[Lon13] M. d. Longueville. "Graph-Coloring Problems." en. In: *A Course in Topological Combinatorics*. Universitext. Springer New York, January 2013, pp. 37–68. ISBN: 978-1-4419-7909-4, 978-1-4419-7910-0 (cited on page 91).

[LZL10] M. Luo, L.-J. Zhang, and F. Lei. "An Insuanrance Model for Guranteeing Service Assurance, Integrity and QoS in Cloud Computing." In: *2010 IEEE International Conference on Web Services (ICWS)*. July 2010, pp. 584–591. DOI: 10.1109/ICWS.2010.113 (cited on page 39).

[MG11] P. Mell and T. Grance. "The NIST Definition of Cloud Computing. National Institute of Standards and Technology." In: *Information Technology Laboratory, Version* 15 (September 2011), 10–7 (cited on pages 23, 24, 27).

[MG95] J. Morrison and J. F. George. "Exploring the software engineering component in MIS research." In: *Communications of the ACM* 38.7 (1995), 80–91 (cited on pages 4, 5).

[Mie08] R. Mietzner. *Using variability descriptors to describe customizable SaaS application templates*. Tech. rep. Technical Report 2008/01, Fakultät Informatik, Universität Stuttgart, 2008 (cited on page 33).

[Mie10] R. Mietzner. "A Method and Implementation to Define and Provision Variable Composite Applications, and its Usage in Cloud Computing." PhD thesis. July 2010 (cited on pages 10, 35).

[MK11] C. Momm and R. Krebs. "A Qualitative Discussion of Different Approaches for Implementing Multi-Tenant SaaS Offerings." In: *Software Engineering (Workshops)*. Vol. 11. 2011 (cited on page 32).

[Mor14] B. Morr. *Untersuchung der Anwendbarkeit von Mixed-Tenancy auf Enterprise SaaS-Applikationen*. Tech. rep. University of Applied Sciences Darmstadt, February 2014, p. 95 (cited on pages 134, 150, 153, 154).

[MUTL09] R. Mietzner, T. Unger, R. Titze, and F. Leymann. "Combining Different Multi-tenancy Patterns in Service-Oriented Applications." In: *Enterprise Distributed Object Computing Conference, 2009. EDOC '09. IEEE International*. 2009, pp. 131–140. ISBN: 1541-7719. DOI: 10.1109/EDOC.2009.13 (cited on page 33).

[Nit09] Nitu. "Configurability in SaaS (software as a service) applications." In: *Proceedings of the 2nd India software engineering conference*. ISEC '09. ACM ID: 1506221. New York, NY, USA: ACM, 2009, 19–26. ISBN: 978-1-60558-426-3. DOI: 10.1145/1506216.1506221 (cited on page 34).

[Okt+12] K. Oktay, V. Khadilkar, B. Hore, M. Kantarcioglu, S. Mehrotra, and B. Thuraisingham. "Risk-Aware Workload Distribution in Hybrid Clouds." In: *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. June 2012, pp. 229–236. DOI: 10.1109/CLOUD.2012.128 (cited on page 38).

[OY13] C. Orou-Yorouba. *Mixed-Tenancy Platform provisioning SaaS-Applications: System Alteration Over Life Cycle*. Tech. rep. University of Applied Sciences Darmstadt - University of Wisconsin Platteville, December 2013 (cited on pages 166, 191, 196, 198, 199).

[PB10] S. Pearson and A. Benameur. "Privacy, Security and Trust Issues Arising from Cloud Computing." In: *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*. November 2010, pp. 693–702. DOI: `10.1109/CloudCom.2010.66` (cited on pages 1, 14).

[PBL10] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. 1st Edition. Springer, November 2010. ISBN: 3642063640 (cited on page 35).

[PKZ11] K. P. N. Puttaswamy, C. Kruegel, and B. Y. Zhao. "Silverline: Toward Data Confidentiality in Storage-intensive Cloud Applications." In: *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. SOCC '11. New York, NY, USA: ACM, 2011, 10:1–10:13. ISBN: 978-1-4503-0976-9. DOI: `10.1145/2038916.2038926` (cited on page 38).

[PP05] D. Pilone and N. Pitman. *UML 2.0 in a Nutshell: A Desktop Quick Reference (In a Nutshell*. 1st ed. O'Reilly Media, June 2005. ISBN: 0596007957 (cited on page 58).

[PTN13] S. Patnaik, P. Tripathy, and K. Naik. *New paradigms in internet computing*. English. Heidelberg; New York: Springer, 2013. ISBN: 9783642354618 3642354610 (cited on page 153).

[RA11] S. T. Ruehl and U. Andelfinger. "Applying software product lines to create customizable software-as-a-service applications." In: ACM Press, August 2011, p. 1. ISBN: 9781450307895. DOI: `10.1145/2019136.2019154` (cited on pages 21, 33).

[RARV12] S. T. Ruehl, U. Andelfinger, A. Rausch, and S. A. W. Verclas. "Toward Realization of Deployment Variability for Software-as-a-Service Applications." In: *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. June 2012, pp. 622 –629. DOI: `10.1109/CLOUD.2012.93` (cited on pages 7, 21).

[Rei+14] M. Reinhardt, S. T. Ruehl, S. A. W. Verclas, U. Andelfinger, and A. Schütte. "Architectural Design of a Deployment Platform to Provision Mixed-tenancy SaaS-Applications." In: *4th International Conference on Cloud Computing and Services Science (CLOSER 2014)*. April 2014 (cited on pages 134, 139).

[Rei13] M. Reinhardt. *Konzeption und Realisierung einer Mixed-Tenancy -fähigen Plattform zur Bereitstellung von SaaS-Applikationen*. Tech. rep. University of Applied Sciences Darmstadt, March 2013 (cited on pages 134, 136, 137, 142).

[RH09] P. Runeson and M. Höst. "Guidelines for conducting and reporting case study research in software engineering." en. In: *Empirical Software Engineering* 14.2 (April 2009), pp. 131–164. ISSN: 1382-3256, 1573-7616. DOI: `10.1007/s10664-008-9102-8` (cited on page 5).

[RN10] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, February 2010. ISBN: 0136042597 (cited on pages 69, 169, 196).

[RTSS09] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. “Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds.” In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS ’09. New York, NY, USA: ACM, 2009, 199–212. ISBN: 9781605588940. DOI: `10.1145/1653662.1653687` (cited on page 38).

[Rue+14] S. T. Ruehl, M. Rupprech, B. Morr, M. Reinhardt, and S. A. W. Verclas. “Mixed-Tenancy in the Wild - Applicability of Mixed-Tenancy for Real-World Enterprise SaaS-Applications.” In: *2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*. July 2014 (cited on page 134).

[Rup13a] M. Rupprech. *Bug #1249355 “XML-RPC object service tries to return non-XML-RP...” : Bugs : OpenERP Server*. November 2013. URL: `https://bugs.launchpad.net/openobject-server/+bug/1249355` (visited on November 9, 2013) (cited on page 157).

[Rup13b] M. Rupprecht. *Einführung von Mixed-Tenancy in eine existierende Enterprise SaaS Applikation durch automatisierte Portierung*. Tech. rep. University of Applied Sciences Darmstadt, August 2013 (cited on pages 134, 145, 147, 148).

[Rus07] T. Russ. *[protege-owl] "Abstract" class in OWL*. September 2007. URL: `https://mailman.stanford.edu/pipermail/protege-owl/2007-September/003823.html` (visited on April 24, 2013) (cited on page 71).

[RWV13] S. T. Ruehl, H. Wache, and S. A. W. Verclas. “Capturing Customers’ Requirements towards Mixed-tenancy Deployments of SaaS-Applications.” In: *2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*. June 2013, pp. 462–470 (cited on pages 7, 43, 69).

[Saa10] G. Saake. *Datenbanken Konzepte und Sprachen*. German. Heidelberg; München; Landsberg; Frechen; Hamburg: mitp, 2010. ISBN: 9783826690570 3826690575 (cited on page 69).

[Sch+12a] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau. “Dynamic Configuration Management of Cloud-based Applications.” In: *Proceedings of the 16th International Software Product Line Conference - Volume 2*. SPLC ’12. New York, NY, USA: ACM, 2012, 171–178. ISBN: 978-1-4503-1095-6. DOI: `10.1145/2364412.2364441` (cited on page 35).

[Sch+12b] J. Schroeter, S. Cech, S. Götz, C. Wilke, and U. Aßmann. “Towards modeling a variable architecture for multi-tenant SaaS-applications.” In: *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*. VaMoS ’12. New York, NY, USA: ACM, 2012, 111–120. ISBN: 9781450310581. DOI: `10.1145/2110147.2110160` (cited on pages 10, 35).

[SES11] A. Schatz, P. Egri, and M. Sauer. “Open Source ERP - Reasonable Tools for Manufacturing SMEs?” In: Fraunhofer Institute for Manufacturing Engineering and Automation IPA, May 2011 (cited on page 144).

[SGB05] M. Svahnberg, J. van Gurp, and J. Bosch. "A taxonomy of variability realization techniques: Research Articles." In: *Software—Practice & Experience* 35 (July 2005). ACM ID: 1070905, 705–754. ISSN: 0038-0644. DOI: 10.1002/spe.v35:8 (cited on page 35).

[Sil+13] C. M. R. da Silva, J. L. C. da Silva, R. B. Rodrigues, L. M. d. Nascimento, and V. C. Garcia. *Systematic Mapping Study On Security Threats in Cloud Computing*. arXiv e-print 1303.6782. (IJCSIS) International Journal of Computer Science and Information Security, Vol. 11, No. 3, March 2013. March 2013 (cited on pages 1, 26, 36, 37, 39).

[SK11] S. Subashini and V. Kavitha. "A survey on security issues in service delivery models of cloud computing." In: *Journal of Network and Computer Applications* 34.1 (January 2011), pp. 1–11. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2010.07.006. URL: http://www.sciencedirect.com/science/article/pii/S1084804510001281 (visited on February 12, 2012) (cited on pages 1, 2, 14, 25, 26).

[SKP13] S. W. Schütz, T. Kude, and K. M. Popp. "The Impact of Software-as-a-Service on Software Ecosystems." In: *Software Business. From Physical Products to Software Services and Solutions*. Springer, January 2013, 130–140 (cited on page 25).

[SLW12] J. Schroeter, M. Lochau, and T. Winkelmann. "Multi-perspectives on Feature Models." In: *Model Driven Engineering Languages and Systems*. Ed. by R. B. France, J. Kazmeier, R. Breu, and C. Atkinson. Lecture Notes in Computer Science 7590. Springer Berlin Heidelberg, January 2012, pp. 252–268. ISBN: 978-3-642-33665-2, 978-3-642-33666-9 (cited on page 35).

[SMS10] Z. Song, J. Molina, and C. Strong. "Trusted Anonymous Execution: A Model to Raise Trust in Cloud." In: *2010 9th International Conference on Grid and Cooperative Computing (GCC)*. November 2010, pp. 133–138. DOI: 10.1109/GCC.2010.37 (cited on page 38).

[SR11] B. Sengupta and A. Roychoudhury. "Engineering multi-tenant software-as-a-service systems." In: *Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems*. PESOS '11. New York, NY, USA: ACM, 2011, 15–21. ISBN: 978-1-4503-0591-4. DOI: 10.1145/1985394.1985397 (cited on pages 1, 2, 32).

[Sri+12] M. K. Srinivasan, K. Sarukesi, P. Rodrigues, M. S. Manoj, and P. Revathy. "State-of-the-art Cloud Computing Security Taxonomies: A Classification of Security Challenges in the Present Cloud Computing Environment." In: *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. ICACCI '12. New York, NY, USA: ACM, 2012, 470–476. ISBN: 978-1-4503-1196-0. DOI: 10.1145/2345396.2345474 (cited on pages 1, 14).

[SS90] A. K. Sethi and S. P. Sethi. "Flexibility in manufacturing: A survey." In: *International Journal of Flexible Manufacturing Systems* 2.4 (July 1990), pp. 289–328. DOI: 10.1007/BF00186471 (cited on page 33).

[Sun+08] W. Sun, X. Zhang, C. J. Guo, P. Sun, and H. Su. "Software as a Service: Configuration and Customization Perspectives." In: *Services Part II, IEEE Congress on*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 18–25. ISBN: 978-0-7695-3313-1. DOI: `http://doi.ieeecomputersociety.org/10.1109/SERVICES-2.2008.29` (cited on page 34).

[SV13] R. Sadykov and F. Vanderbeck. "Bin Packing with Conflicts: A Generic Branch-and-Price Algorithm." en. In: *INFORMS Journal on Computing* 25.2 (March 2013), pp. 244–255. ISSN: 1091-9856, 1526-5528. DOI: `10.1287/ijoc.1120.0499` (cited on page 204).

[SVD03] K. Satyendra, R. V. Venkata, and T. Devanath. *A heuristic procedure for one dimensional bin packing problem with additional constraints*. Tech. rep. Indian Institute of Management Ahmedabad, Research and Publication Department, 2003 (cited on pages 92, 204).

[Szy02] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. 2nd ed. Addison-Wesley Professional, November 2002. ISBN: 0201745720 (cited on page 11).

[WA11] K. Wood and M. Anderson. "Understanding the Complexity Surrounding Multitenancy in Cloud Computing." In: *2011 IEEE 8th International Conference on e-Business Engineering (ICEBE)*. 2011, pp. 119–124. DOI: `10.1109/ICEBE.2011.68` (cited on page 1).

[Wai08] P. Wainewright. *Many degrees of multi-tenancy*. June 2008. URL: `http://www.zdnet.com/blog/saas/many-degrees-of-multi-tenancy/533` (visited on October 3, 2013) (cited on page 33).

[Wal07] W. D. Wallis. *Beginner's Guide to Graph Theory, Second Edition*. English. [S.l.]: Birkhäuser Boston, 2007. ISBN: 9780817644840 0817644849 (cited on pages 44, 89).

[Wan+11] R. Wang, Y. Zhang, S. Liu, L. Wu, and X. Meng. "A Dependency-Aware Hierarchical Service Model for SaaS and Cloud Services." In: *2011 IEEE International Conference on Services Computing (SCC)*. July 2011, pp. 480–487. DOI: `10.1109/SCC.2011.17` (cited on page 35).

[Wan06] R. Wanka. *Approximationsalgorithmen eine Einführung*. German. Wiesbaden: B.G. Teubner Verlag / GWV Fachverlage, Wiesbaden, 2006. ISBN: 97838351-90672 3835190679 (cited on pages 19, 90, 91, 117).

[Weg03] I. Wegener. *Komplexitatstheorie*. de. Springer DE, March 2003. ISBN: 978354-0001614 (cited on page 94).

[Wil02] R. J. Wilson. *Four colors suffice: how the map problem was solved*. English. Princeton, NJ: Princeton University Press, 2002. ISBN: 0691115338 978069-1115337 0691120234 9780691120232 (cited on page 92).

[WP67] D. J. A. Welsh and M. B. Powell. "An upper bound for the chromatic number of a graph and its application to timetabling problems." en. In: *The Computer Journal* 10.1 (January 1967), pp. 85–86. ISSN: 0010-4620, 1460-2067. DOI: `10.1093/comjnl/10.1.85` (cited on page 93).

[Zha+07] K. Zhang, X. Zhang, W. Sun, H. Liang, Y. Huang, L. Zeng, and X. Liu. "A Policy-Driven Approach for Software-as-Services Customization." In: *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*. 2007, pp. 123–130. DOI: `10.1109/CEC-EEE.2007.9` (cited on page 35).

[Zha+12] O. Zhang, R. Ko, M. Kirchberg, C. H. Suen, P. Jagadpramana, and B. S. Lee. "How to Track Your Data: Rule-Based Data Provenance Tracing Algorithms." In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. June 2012, pp. 1429–1437. DOI: `10.1109/TrustCom.2012.175` (cited on page 39).

[ZJRR12] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. "Cross-VM side channels and their use to extract private keys." In: *Proceedings of the 2012 ACM conference on Computer and communications security*. CCS '12. New York, NY, USA: ACM, 2012, 305–316. ISBN: 978-1-4503-1651-4. DOI: `10.1145/2382196.2382230` (cited on page 26).

[ZSLB09] K. Zhang, Y. Shi, Q. Li, and J. Bian. "Data Privacy Preserving Mechanism Based on Tenant Customization for SaaS." In: *International Conference on Multimedia Information Networking and Security, 2009. MINES '09*. Vol. 1. 2009, pp. 599–603. DOI: `10.1109/MINES.2009.256` (cited on pages 14, 25).

[Ama13] Amazon. *Amazon Elastic Compute Cloud (Amazon EC2)*. October 2013. URL: `http://aws.amazon.com/de/ec2/` (visited on April 10, 2010) (cited on page 24).

[Apa13] Apache Foundation. *Apache Jena*. March 2013. URL: `http://jena.apache.org/` (visited on April 11, 2013) (cited on page 70).

[Apa14] Apache OFBiz. *Apache OFBiz, The Apache Open For Business Project - Open Source E-Business / E-Commerce, ERP, CRM, POS, SCM, MRP, CMMS/EAM*. 2014. URL: `http://ofbiz.apache.org/` (visited on April 18, 2014) (cited on page 144).

[Clo13a] Cloud Security Alliance. *About : Cloud Security Alliance*. June 2013. URL: `https://cloudsecurityalliance.org/about/` (visited on July 6, 2013) (cited on page 25).

[Clo13b] Cloud Security Alliance. *The Notorious Nine: Cloud Computing Top Threats in 2013*. Tech. rep. February 2013. URL: `https://cloudsecurityalliance.org/download/the-notorious-nine-cloud-computing-top-threats-in-2013/` (visited on April 30, 2013) (cited on pages 1, 14, 25).

[Com14] Compiere Open Source ERP. *Compiere Open Source ERP - A Modern, Low-cost ERP Software Solution*. 2014. URL: `http://www.compiere.com/` (visited on April 18, 2014) (cited on page 144).

[Goo13] Google. *Google App Engine - Google Code*. October 2013. URL: `http://code.google.com/intl/en-NE/appengine/` (visited on April 10, 2010) (cited on page 24).

[Kiv14] Kivitendo. *Kivitendo: kivitendo*. 2014. URL: `http://www.kivitendo.de/` (visited on April 18, 2014) (cited on page 144).

[Lim14] Limbas. *Limbas Wiki*. 2014. URL: `http://www.limbas.org/wiki/Hauptseite` (visited on April 18, 2014) (cited on page 144).

[Ope14a] Openbravo. *Openbravo, the preferred Commerce and ERP Platform*. 2014. URL: `http://www.openbravo.com/` (visited on April 18, 2014) (cited on page 144).

[Ope14b] Opentaps. *Opentaps*. 2014. URL: `http://www.opentaps.org/` (visited on April 18, 2014) (cited on page 144).

[Ops13] Opscode. *Chef | Opscode*. November 2013. URL: `http://www.opscode.com/chef/` (visited on November 14, 2013) (cited on page 153).

[Pup13] Puppet Labs. *Puppet Labs: IT Automation Software for System Administrators*. November 2013. URL: `http://puppetlabs.com/` (visited on November 14, 2013) (cited on page 153).

[The13] The GraphStream Team. *GraphStream - A Dynamic Graph Library*. October 2013. URL: `http://graphstream-project.org/` (visited on October 6, 2013) (cited on page 70).

[Try14] Tryton. *Tryton*. 2014. URL: `http://www.tryton.org/de/` (visited on April 18, 2014) (cited on page 144).

[hei09] heise open. *Quelloffene Kür: Open-Source-ERP-Systeme im Vergleich*. Komplexe Geschäftsanwendungen auf Open-Source-Basis sind in Deutschland eine Ausnahmeerscheinung – dennoch gibt es sie. Sechs der bekannteren ERP-Systeme können als ernstzunehmende Kandidaten gelten. September 2009. URL: `http://www.heise.de/open/artikel/Quelloffene-Kuer-Open-Source-ERP-Systeme-im-Vergleich-763963.html` (visited on November 3, 2013) (cited on page 144).

[pro13] protégé team. *The Protégé Ontology Editor and Knowledge Acquisition System*. October 2013. URL: `http://protege.stanford.edu/` (visited on October 6, 2013) (cited on page 70).

[sal13] salesforce. *CRM - salesforce.com*. October 2013. URL: `http://www.salesforce.com/` (visited on April 10, 2010) (cited on page 24).

[ADe14] ADempiere. *ADempiere*. 2014. URL: `http://www.adempiere.com` (visited on April 18, 2014) (cited on page 144).

[CAO14] CAO-Faktura. *CAO-Faktura Warenwirtschaftsystem | CAO-Faktura*. 2014. URL: `http://www.cao-wawi.de/` (visited on April 18, 2014) (cited on page 144).

[Inf12] InfoWorld. *Bossie Awards 2012: The Best of Open Source Software Awards*. September 2012. URL: `http://www.infoworld.com/d/open-source-software/bossies-2012-the-best-of-open-source-software-awards-202465` (visited on November 3, 2013) (cited on page 145).

[Inf13] InfoWorld. *Bossie Awards 2013: The best open source applications - InfoWorld*. September 2013. URL: `http://www.infoworld.com/slideshow/119652/bossie-awards-2013-the-best-open-source-applications-226975` (visited on November 16, 2013) (cited on page 145).

[OMG13a] OMG. *OCL*. April 2013. URL: `http://www.omg.org/spec/OCL/` (visited on April 11, 2013) (cited on page 59).

[OMG13b] OMG. *UML*. April 2013. URL: `http://www.omg.org/spec/UML/` (visited on April 11, 2013) (cited on page 58).

[OWA13a] OWASP. *Category:OWASP Cloud - 10 Project - OWASP*. June 2013. URL: `https://www.owasp.org/index.php/Category:OWASP_Cloud_-_10_Project` (visited on June 29, 2013) (cited on pages 1, 26).

[OWA13b] OWASP. *Cloud-10 Multi Tenancy and Physical Security - OWASP*. June 2013. URL: `https://www.owasp.org/index.php/Cloud-10_Multi_Tenancy_and_Physical_Security` (visited on June 29, 2013) (cited on page 26).

[Ope13] OpenERP. *OpenERP descripiton of Apps*. November 2013. URL: `https://www.openerp.com/apps/` (visited on November 17, 2013) (cited on page 171).

[Ope14] OpenERP. *OpenERP - Beautiful Business Applications*. March 2014. URL: `https://www.openerp.com/` (visited on March 13, 2014) (cited on pages 144, 145, 171).

[SQL14] SQL-Ledger ERP. *SQL-Ledger ERP*. 2014. URL: `http://www.sql-ledger.com/` (visited on April 18, 2014) (cited on page 144).

[SYN14] SYNERPY GmbH. *Open Source ERP AvERP - Warenwirtschaftssystem der SYNERPY GmbH*. 2014. URL: `http://www.synerpy.de/cm/` (visited on April 18, 2014) (cited on page 144).

[W3C04] W3C. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. May 2004. URL: `http://www.w3.org/Submission/SWRL/` (visited on October 6, 2013) (cited on page 70).

[W3C12] W3C. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. December 2012. URL: `http://www.w3.org/TR/owl2-overview/` (visited on April 11, 2013) (cited on page 69).

[W3C13] W3C. *SPARQL 1.1 Query Language*. March 2013. URL: `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/` (visited on April 19, 2013) (cited on page 70).

[web14] webERP. *webERP Home*. 2014. URL: `http://www.weberp.org/` (visited on April 18, 2014) (cited on page 144).

[xTu14] xTuple. *xTuple | Open Source ERP for Mac, Linux and Windows*. 2014. URL: `https://www.xtuple.com/` (visited on April 18, 2014) (cited on page 144).