Who may share with Whom?!
Mixed-Tenancy: Conditional Sharing of Cloud Resources

Stefan T. Ruehl[1], Urs Andelfinger[2], Andreas Rausch[1], and Stephan A.W. Verclas[3]
[1]Clausthal University of Technology
Albrecht-von-Groddeck-Str. 7, 38678 Clausthal-Zellerfeld, Germany
Email: {stefan.t.ruehl, andreas.rausch}@tu-clausthal.de
[2]University of Applied Sciences Darmstadt
Haardtring 100, 64295 Darmstadt, Germany
Email: urs.andelfinger@h-da.de
[3]T-Systems International GmbH
Heinrich-Hertz-Str. 1, 64295 Darmstadt, Germany
Email: stephan.verclas@t-systems.com

Abstract: Software-as-a-Service is a delivery model whose basic idea is to provide applications to the customer on demand over the Internet. SaaS thereby promotes multi-tenancy as a tool to exploit economies of scale. This means that a single application instance serves multiple customers. However, a major drawback of SaaS is the customers' hesitation of sharing infrastructure, application code, or data with other tenants. 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. So far the only approach in research to counteract on this hesitation has been to enhance the isolation between tenants using the same instance. Our approach (presented in earlier work) tackles this hesitation differently. It allows customers to choose if or even with whom they want to share the application. The approach enables the customer to define their constraints for individual application components and the underlying infrastructure. Based on these constraints, the SaaS application is deployment in the way that all constraints are satisfied and only minimal resources are required.

This paper's contribution is a summary of all results that were developed during the creation of the mixed-tenancy approach. These results focus on the challenges of capturing customers' deployment constraints, computing a valid and optimal deployment, and an evaluation of the general applicability of mixed-tenancy in real-world.

1 Introduction – Problem Analysis

Software-as-a-Service (SaaS) is a delivery model whose basic idea is to provide applications to the customer on demand over the Internet. In contrast to similar but older approaches, SaaS promotes multi-tenancy as a tool to exploit economies of scale. Multi-tenancy means that a single application instance serves multiple customers at the same time. However, even though multiple customers use the same instance, each of them has the impression that the instance is designated only to themselves. This is achieved by isolating the tenants' data from each other [CC06].

In contrast to single-tenancy, multi-tenancy has the advantage that IT infrastructure may be used as efficiently as possible to host as many tenants as possible on the same instance. Thus, operational and maintenance cost of the application is decreased. However, one of the major threats of multi-tenancy applications is information disclosure due to data breach [Clo13]. This may occur through a system error, malfunction, or destructive actions. So far this problem has only been tackled by proposing new approaches to implement and improve the tenants' isolation on a single instance (e.g. [GSH+07], [BAT12], and [AGI12]).

Our approach presented in previous works ([RARV12], [RWV13], [LMRV14], [RRM+14], [Rue14]) is different, however, as it strives to solve the problem by finding a hybrid solution between multi-tenancy and single-tenancy. We refer to this approach as *mixed-tenancy* (MT). The approach tries to emphasize both, the customers' concerns or fear about sharing infrastructure as well as the operator's desire to utilize infrastructure as efficiently as possible in order to gain low cost.

The basic idea of the MT is to utilize component-based applications. These so-called composite SaaS-applications are composed of a number of application components (AC) that each offer atomic functionality. Based on these applications, the MT approach enables the customer to provide constraints that state if or even with whom they are willing to share. They may express their constraints not just for the ACs, but also for the underlying infrastructure stack an application component utilizes (e.g. application server, virtual machine). Based on these constraints, a deployment is computed that complies to all constraints (it is valid) and, thereby, only utilizes minimal resources (it is optimal). The computed deployment is afterwards used as a blueprint according to which the SaaS-application is deployed.

When developing the MT approach the following three research questions were identified to be important and, thus, were tackled first. ***Research Question 1 (RQ-1)*** – How can customers' constraints towards MT be described? ***Research Question 2 (RQ-2)*** – How can a valid and optimal deployment be found in polynomial time? ***Research Question 3 (RQ-3)*** – Is it possible to apply the MT approach to existing composite applications? In this paper we present a summary of the results related to these research questions. Thus, the rest of this paper structures accordingly. Section 2 discusses the work done with respect to research question RQ-1 and summarizes the results achieved. For research question RQ-2 the same is done by Section 3. And finally, the results produced for research question RQ-3 are discussed in Section 4. Section 5 will conclude this paper by summarizing its results and outlining opportunities to develop the MT approach further.

2 Capturing Customers Deployment Constraints

The first research question to be tackled in order to create the MT approach was RQ-1. It deals with capturing customers' deployment constraints towards MT. The special challenge involved is that it shall be possible for customers to express with which other tenants they do and do not want to share ACs (and the underlying infrastructure), even if the operator does not reveal its customer base to the customers. This section will provide the

description of a generic description model that allows that and, further, may be used for a wide variety of different applications.

2.1 Conceptual Analysis Based on Requirements

The first step towards the definition of the description model is to analyze what needs to be captured. The following is a summary.

Deployment Levels (DL) – It has previously been stated (Section 1) that customers shall not just be able to express their deployment constraints for the ACs but also for the underlying infrastructure (e.g. application server, virtual machine). Within this work we refer to those slices of the stack for which constraints shall be expressible as DLs. However, in order to have a slice serve as a DL, it is necessary that they fulfill certain characteristics, like that it is possible to instantiate multiple units of a single DL and that any unit of a DL may host multiple of a higher DL[1].

Deployment Models (DM) – It is the idea of this model to allow customers to express different types of deployment constraints. These types are called DMs and differ in the way how customers may state the tenants they may share with. The following five are those considered: *private* (no sharing shall be allowed), *public* (sharing is allowed with all other tenants), *white hybrid* (sharing shall only be possible with specific tenants that are included – e.g. only with companies from Europe), *black hybrid* (sharing shall be possible with all tenants but with those that are excluded – e.g. not with competitors), and *gray hybrid* (sharing shall be possible only with specific tenants but not with specific others – e.g. only with companies from Europe but not with competitors).

Groups – In order to be able to express constraints like the ones just mentioned, it is necessary to have means of categorizing tenants. Thus, a group is an entity to which all tenants are associated that share the commonality the group represents (e.g. geographic regions like Europe, USA).

Dimensions – Further, it shall be possible to create groups that belong to different topics, such as industries (containing different industries e.g. finance, IT, telecommunications), geographic (containing the countries of geographic regions tenants operate in), or other things like data privacy acts tenants may apply to. Within this work, these topics are referred to as dimensions. Each dimension is realized by multiple groups that belong to only this dimension. Further, groups may have a subset relationship between them in the shape of an acyclic graph. This allows tenants to more precisely express their constraints since they may, for example, choose to exclude banks but not insurances, or exclude all customers from the entire finance industry (thereby excluding both banks and insurances).

Virtual Tenants – In the introduction to this section, it has been stated that keeping customer base secret to the customers, is one of the major objectives of the description model. The concept of virtual tenants allows to define explicitly which tenants shall be included or excluded. They represent companies that exist in the market but are not customers of

[1] A full list of requirements may be found in [RWV13] and [Rue14].

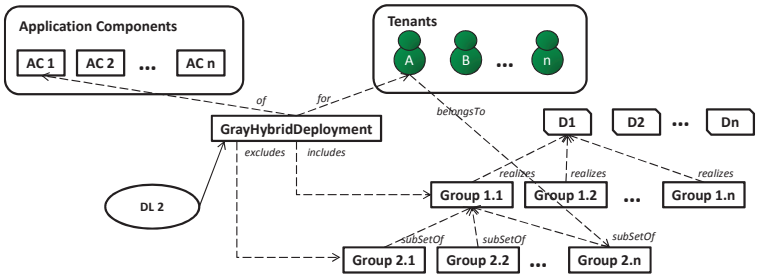

Figure 1: Description of a Deployment Constraint

the application yet. If one tenant demands to exclude another specific tenant, for example, but this tenant has not subscribed yet, it is possible to use virtual tenants as dummies to express constraints but they will not be considered when computing a deployment.

In order to verify that, the requirements just presented are sufficient to be used in reality. They have been discussed with several experts from both academia and industry.

2.2 Process of Deployment Constraint Definition

Due to the fact that the description model 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 following introduces the necessary steps to capture customers' deployment constraints. *Step 1 – Customization* This step's purpose is to adapt the generic model to the specific application for which deployment constraints shall be captured. Thus, it covers the capturing of the specific ACs, DLs, dimensions, groups, and tenants. *Step 2 – Grouping* This step's purpose is to associate tenants to those groups they belong to (each tenant needs to be assigned to at least one group per dimension). 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. Thus, this step is crucial since if data is flawed, deployment constraints defined may not be properly realized. Again, these first two steps are executed by the operator. *Step 3 – Constraint Definition* Finally, in this step customers express their deployment constraints. Figure 1 illustrates an example for this. In the example tenant A chooses a *gray hybrid* DM to express that they want to share units of DL 2 for AC 1 only with tenants that belong to group 1.1 but not with tenants belonging to group 2.1. This means sharing will only be possible with tenants that belong to the groups 1.1, 2.2, ..., 2.n. *Step 4 – Deployment Information (DI) Extraction* The final step of the process is to extract the information about which tenants may share which instances of ACs and units of DLs, called DI. For a specific combination of AC and DL it is possible to represent this information as an undirected graph. In such a graph the vertices represent the tenants and an edge between two vertices states that those tenants are allowed to share. In order to comply with all constraints given by customers, there shall only be an edge between two tenants if none of the tenants has stated a constraint that prohibits

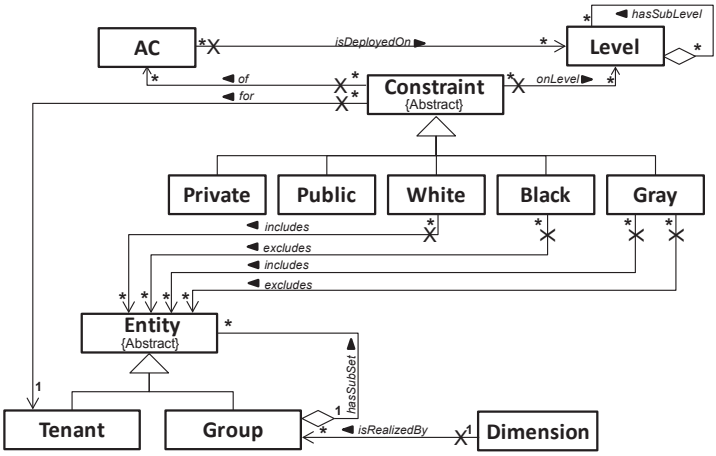

Figure 2: Description Model as UML Class Diagram (incomplete)

sharing. Thus, DI is a collection of graphs, one for every combination of an application component and a deployment level (where the application component is deployed on the deployment level). An example of it is illustrated on the left hand side of Figure 3.

2.3 Introduction of Description Model

Based on the previous two subsections, it is possible to introduce the description model. Figure 2 gives an overview of it using UML class diagram notation. It illustrates the entities that were discussed in previous sections as classes. Furthermore, there are relations between these classes that may be used to describe deployment constraints. In Figure 1 an example was given in which tenant A defined that they want to use AC-1 as a gray deployment on DL-2. Realizing this based on the UML Model would mean that there is an object of the type AC, called *AC-1*. In addition, there is an object *A* of type tenant and an object *DL-2* of type Level. These objects would have been created by the operator as part of the first step.

In order to create the aforementioned deployment constraint, they would create an object of type public. This object is then associated to *AC-1* by using the *of*-relationship, to *A* by using the *for*-relationship, to *DL-2* by using the *onLevel*-relationship, to group 1.1 by the *include*-relationship, and to group 2.1 by the *exclude*-relationship. Thus, the constraint is captured.

Please note that the structure between tenants, groups and dimensions 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.

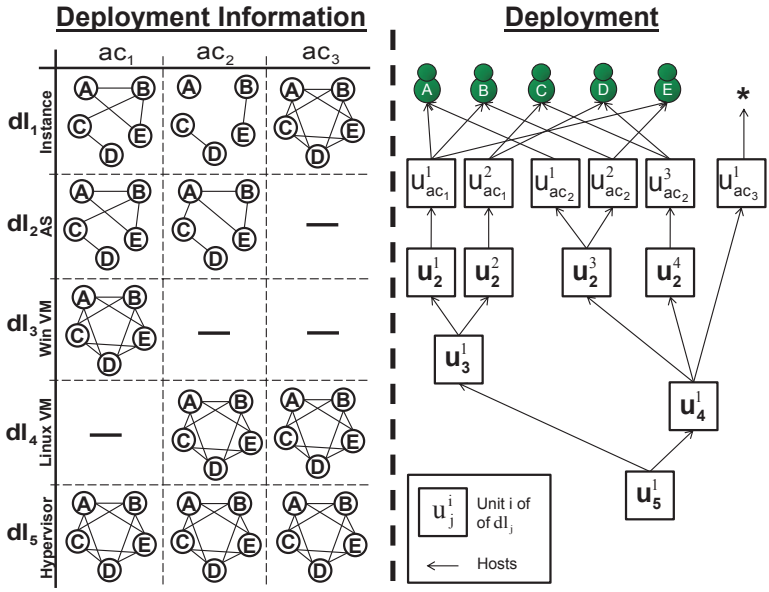

Figure 3: An Example of a Valid and Optimal Deployment

A realization of the model was created by utilizing the Web Ontology Language (OWL)[2]. Furthermore, it was evaluated if the model is actually able of providing the expressiveness required to model all requirements discussed. Based on an example that consisted of all cases, it was possible to show that the expressiveness of the model is sufficient. Details on both, the realization and the evaluation may be found in [RWV13] and [Rue14].

3 Computation of a Valid and Optimal Deployment

The second research question deals with the computation of a valid and optimal deployment. This question is addressed in this section.

3.1 Problem Description

The first step in this section is to introduce the deployment problem. This is done based on the example that is illustrated by Figure 3 (left hand side).

The input for the problem is all constraints given by customers in an aggregated from. It consists of multiple undirected graphs, one for every combination of AC and DL, where the AC shall be deployed on the DL. Each graph contains all tenants as vertices. An edge

[2] http://www.w3.org/TR/owl2-overview/

between two vertices states that these two tenants are allowed to share an AC on a given DL. It covers three ACs and five DLs. As visible, not all ACs are deployed on all DLs.

Based on the input, it is the goal to compute a deployment that is valid and optimal. A deployment shall be a collection of units for every DL. On the highest level units need to be created for every AC separately since each one needs to be instantiated (e.g. ac_1: $u^1_{ac_1}, u^2_{ac_1}$). On the other DLs only one type of unit needs to be created per DL (e.g. dl_2: $u^1_2, u^2_2, u^3_2, u^4_2$). Furthermore, units of the highest DL hosts tenants directly, the other units host units of higher DLs. A valid and optimal deployment for the example is illustrated on the right hand side of Figure 3. For each of the ACs, units were created. Each of them hosts tenants.

Due to the structure of a deployment, it is not possible to assign any tenants to the same units of the highest level but only those that are allowed to do so according to the related graphs. This is also considered in the example (e.g. $u^1_{ac_1}$ is shared by tenants A, B, and E which are all adjacent to each other in the corresponding graph). Units of the DLs dl_2 – dl_5 are used transitively through the host's relationship (e.g. all tenants use u^1_5). For these units there are two constraints that need to be considered in order to have two units share the same lower unit. The first one is that the DI must permit it. For this all graphs need to be considered that express the constraints for the evolved ACs (e.g. $u^1_{ac_4}$ is transitively used by all tenants and it deploys units from dl_2 and dl_3. Thus, sharing must be permitted between all tenants in the graphs of ac_2 and ac_3 on dl_4 - this is true since they are complete). The second requirement that needs to be fulfilled is that units of different ACs may only be hosted by the same lower unit if they share the entire lower DL stack. Based on the example, it is possible to describe why this is necessary. If this constraint was not considered, it would be possible to find a set of dl_2 units that host all dl_1 units (one for $u^1_{ac_1}, u^2_{ac_1}, u^2_{ac_2}$ and one for $u^1_{ac_2}, u^3_{ac_2}$). However, these units would not be deployable on the Levels dl_2 and dl_3 since they deploy instances of both ac_1 and ac_2. This would lead to an invalid deployment.

All this that has just been discussed needs to be fulfilled in order to have a valid deployment. However, in order to have a valid and optimal deployment the deployment needs to cause only minimal cost.

3.2 Analysis of the Deployment Problem

The first analysis that can easily be done is determining the complexity of the deployment problem. This can be done by reducing the clique cover problem to the problem at hand. The clique cover problem is an NP-hard graph theory problem whose goal is to split a given graph into a minimal number of disjoint cliques (complete subgraphs) [Kar72]. For the problem reduction, let's consider a problem instance of the deployment problem where there exists only one AC and one DL. For this, the DI would consist of only one graph. A valid and optimal deployment would require a minimal number of units that each hosts a set of tenants that are a clique in the graph (may share with each other). Therefore, the problem is equivalent to clique cover and, thus, is NP-hard.

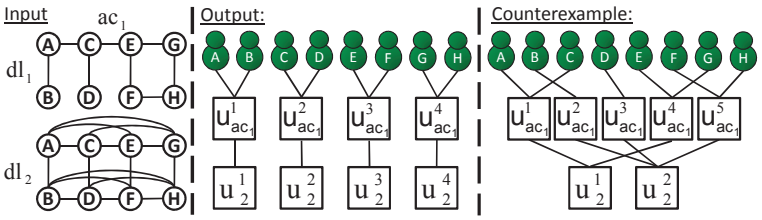

Figure 4: Example not utilizing a minimal Clique Cover on highest Level

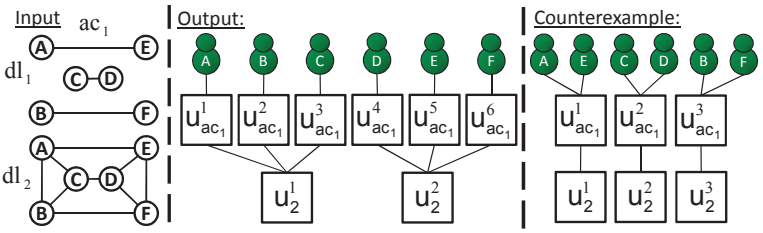

Figure 5: Example not utilizing a minimal Clique Cover on lowest Level

Based on the knowledge that clique cover is related to the deployment problem, an obvious assumption would be that finding a minimal clique cover on all DLs will allow computing a valid and optimal deployment. However, there are counter examples for this assumption.

Assuming that the cost of each instance is one, Figure 4 illustrates an example where a deployment utilizing a non-minimal clique cover on the highest DL causes less cost than one utilizing a minimal clique cover. Figure 5 illustrates the same for the lowest DL. For this example, choosing a minimal clique cover on the lowest DL will increase the cost. Furthermore, when combining the two examples, it is possible to create an example where a valid and optimal deployment will not utilize a minimal clique cover at all.

3.3 Introduction of Intuitive Algorithms

The conclusion gained from the previous subsection is that computing a minimal clique cover will not always allow computing a valid and optimal deployment. However, there are many algorithms available to approximate a valid and optimal deployment in polynomial time. Thus, we developed the following two intuitive algorithms: *Top-Down* – A valid deployment is approximating, by approximating a minimal clique cover for the highest DL. Based on this, the algorithm works its way down, always approximating a minimal clique cover by considering the higher clique cover. It was possible to prove that the relative performance guarantee of this approach is not better than 2. *Bottom-Up* – A valid deployment is approximating the other way around starting from bottom working its way up. For this approach it was possible to prove that its relative performance guarantee is not better than $3/2$. Details on both algorithms may be found in [LMRV14] and [Rue14].

In addition to the theoretical results, we also conducted an experimental evaluation. For this a total of 2.332.800 problem instances were randomly generated and both algorithms were used to compute a valid and optimal deployment. For all problem instances it was assumed that they follow the structure illustrated by Figure 3 (having five DLs and three ACs). However, each problem instance contained 50 tenants. Their deployment constraints were created randomly[3]. The experiments show that for about 67% of all problem instances the top-down approach outperforms the bottom-up approach (with an average improvement of 2.4%). In about 17% of the cases the results were equal. Thus, results suggest to prefer the top-down approach.

So far it has been assumed that all units cause cost of one. However, since this is probably not applicable in reality, we also created an algorithm that is capable of computing a valid deployment in any given order of DLs (for example from the most to the least expensive). Results for this approach are very promising and suggest that this approach may be beneficial. A detailed analysis of that is, however, still open.

4 Case Study: ERP-System as Mixed-Tenancy Cloud Service

The last research question introduced in the introduction of this paper was RQ-3. It deals with analyzing real-world applicability of the MT approach. In order to investigate this, a case study was conducted where MT has been introduced to an existing real-world application.

4.1 Deployment Platform

The first step towards deploying an existing application is to do the conceptual design of a deployment platform (DP). This platform has basically two primary jobs. The first challenge is deploying an application according a computed deployment. Realizing a platform that is capable of doing this for a specific application is straight forward using tools like Chef[4]. The second challenge is establishing communication between different instances of ACs. This is more difficult since it requires first of all that communication is decentralized and that platform logic is separated. *Decentralized Communication* – The goal of MT is to allow customers to express deployment constraints that restrict with whom they share resources. Following this idea it is necessary that the constraints are also enforced for the communication between AC instances. This means that tenants that do not wish to share resources shall also not be able share the same platform components that establish communication. This may be realized by creating a communication mechanism that is distributed without having a central system used by all tenants. *Separation of Platform Logic* – It is the aim of research question RQ-3 to evaluate if the MT approach is applicable to existing applications. Thus, the DP must have the ability to deploy existing applications according

[3] Details of the experiments may be found in [Rue14].

[4] `http://www.getchef.com/`

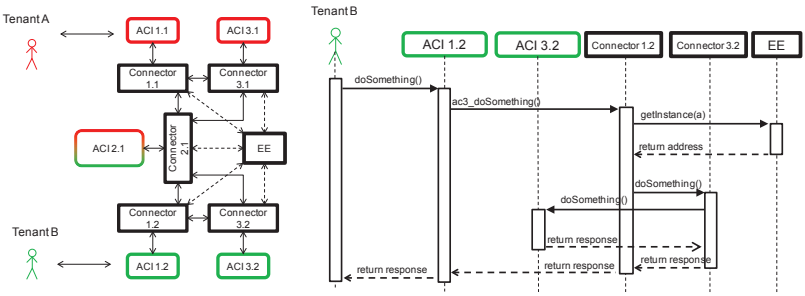

Figure 6: Overview of the Connector Pattern

to the MT paradigm. In order to have minimal effort needed to migrate an existing application onto the DP, it would be beneficial 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.

In order to tackle this challenge, a communication pattern was created that is illustrated by Figure 6. Its basic idea is to realize the separation of platform logic by delegating all communication of an instance through an adapter. Depending of the tenant that is triggering communication, this adapter is capable of directing a request to the right instance.

4.2 Case Study: OpenERP as Mixed-Tenancy Application

In order to evaluate applicability of MT to real-world applications we analyzed applying it to the open source ERP-system OpenERP. The MT approach requires that ACs that compose an application may be deployed separately. This is due to the fact that there may be multiple instances of the same AC that are used by different tenants. Unfortunately, we had to find out that OpenERP's architecture does not allow deploying individual modules separately. Thus, it was necessary to split OpenERP into ACs that are separately deployable. This increased the resource demand of the application significantly. However, based on this version of OpenERP, it was possible to implement a DP that allows deploying and using OpenERP following the MT approach realizing the concepts introduced in the previous subsection.

Further, we evaluated that the deployment of OpenERP following MT behaves exactly like a regular deployed version by creating 151 test cases. Based on these, it can be concluded that it is possible to successfully introduce the MT approach to existing real-world applications.

As stated before, for OpenERP it was necessary to alter the architecture. This increased resource demand was so significant that the created MT deployment required more resources that a single-tenancy deployment of the original version of OpenERP. This implies that only those applications may beneficially be used for the MT approach where deploying ACs multiple times causes no or only minimal overhead. For applications where this is

not possible, it is still possible to describe the entire application as one single AC and have customers express constraints for sharing of the entire application. Thus, resource would be used more efficiently than in single-tenancy, any time there is sharing possible[5].

5 Conclusion

This paper summarized the current state of the development of the MT approach. Therefore, it addressed some of the major challenges involved.

It described a description model that allows customers to express their deployment constraints about if or with whom they are willing to share specific ACs on specific DLs. This model was designed to be very generic in order to be reusable for a wide variety of composite multi-tenancy applications. Furthermore, it allows operators to keep their customer base secret. Based on all deployment constraints that are captured, it is possible to extract the aggregation information about which tenants are allowed to share which ACs on which DL. This is the input for the algorithm that computes 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. It was proven that the problem of computing such a 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 undertaken so far was an evaluation of the applicability of the MT 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 MT 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.

Besides the results created so far, MT still offers many opportunities for future research. One significant opportunity is the creation of a sufficient approach to ensure that tenants are correctly associated to groups. This approach would be much needed because flawed data would results in constraints not being realized correctly (e.g. European company is associated to Asian group). Secondly, additional research can be done on finding a better algorithm to compute a valid and optimal deployment. So far only the two intuitive

[5] A full discussion of the case study may be found in [RRM+14] and [Rue14].

approaches were developed. Furthermore, another big point is that so far only the initial deployment of a MT application has been investigated. However, it is very likely that customers' deployment constraints will change over time and the application's deployment needs to be adapted.

References

[AGI12] M. Almorsy, J. Grundy, and A.S. Ibrahim. TOSSMA: A Tenant-Oriented SaaS Security Management Architecture. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 981–988, 2012.

[BAT12] W.J. Brown, V. Anderson, and Qing Tan. Multitenancy - Security Risks and Countermeasures. In *2012 15th International Conference on Network-Based Information Systems (NBiS)*, pages 7–13, 2012.

[CC06] Frederick Chong and Gianpaolo Carraro. Architecture Strategies for Catching the Long Tail. *Microsoft MSDN*, April 2006.

[Clo13] Cloud Security Alliance. The Notorious Nine: Cloud Computing Top Threats in 2013. Technical report, February 2013.

[GHJ94] Erich Gamma, Richard Helm, and Ralph E. Johnson. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, Amsterdam, 1st ed., reprint. edition, October 1994.

[GSH+07] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo 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)*, pages 551–558, Tokyo, Japan, July 2007.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. Technical report, Springer, 1972.

[LMRV14] Steffen Lange, Marian Margraf, Stefan T. Ruehl, and Stephan 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.

[RARV12] Stefan T. Ruehl, Urs Andelfinger, Andreas Rausch, and Stephan A. W. Verclas. Toward Realization of Deployment Variability for Software-as-a-Service Applications. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 622 –629, June 2012.

[RRM+14] Stefan T. Ruehl, Malte Rupprech, Bjorn Morr, Matthias Reinhardt, and Stephan 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.

[Rue14] Stefan T. Ruehl. *Mixed-Tenancy Systems - A hybrid Approach between Single and Multi-Tenancy*. PhD thesis, Clausthal University of Technology, 2014.

[RWV13] Stefan T. Ruehl, Holger Wache, and Stephan A. W. Verclas. Capturing Customers' Requirements towards Mixed-tenancy Deployments of SaaS-Applications. In *2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*, pages 462–470, June 2013.