Capturing Customers' Requirements towards Mixed-tenancy Deployments of SaaS-Applications

Stefan T. Ruehl
Clausthal University of Technology
Albrecht-von-Groddeck-Str. 7
38678 Clausthal-Zellerfeld, Germany
stefan.t.ruehl@tu-clausthal.de

Holger Wache
University of Applied Sciences
Northwestern Switzerland
Riggenbachstrasse 16
4600 Olten, Switzerland
holger.wache@fhnw.ch

Stephan A. W. Verclas
T-Systems International GmbH
Heinrich-Hertz-Str. 1
64295 Darmstadt, Germany
stephan.verclas@t-systems.com

***Abstract*—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. This means that a single application instance serves multiple customers.**
However, a major throwback 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 by individual users. 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 make that choice not just for the entire application but specifically for individual application components (AC). One of the challenges towards the realization of a mixed-tenancy platform is the capturing of customer requirements. This is why the focus of this paper is to present an approach that enables tenants to express their unique deployment needs. This approach is realized using Semantic Web technologies.

***Keywords*-Software architecture; Software design; Web and internet services;**

I. Introduction

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 (MT) as a tool to exploit economies of scale. This means that a single application instance serves multiple customers. However, even though multiple customers use the same instance each of them has the impression that the instance is designated only to them. This is achieved by isolating the tenants' data from each other [1].
In contrast to single-tenancy, MT has the advantage that IT-Infrastructure may be used most efficiently as it is 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 MT Applications is information disclosure due to data breach [2]. This may occur through a system error, malfunction, or destructive action. So far this problem has only been tackled by proposing new approaches to implement and improve the tenant isolation on a single instance.

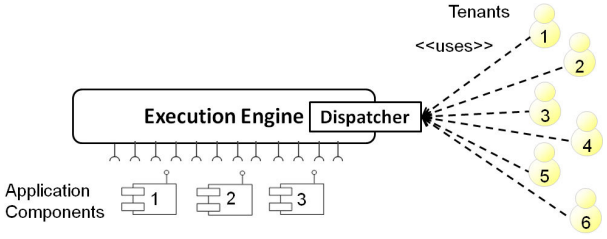

Figure 1. Architectural Overview

The approach of this work, however, is different as it strives to solve the problem by finding a hybrid solution between multi-tenancy and single-tenancy, that we call *mixed-tenancy*. The approach tries to emphasis both the customers' concerns about sharing infrastructure as well as the operator's desire to utilize infrastructure as efficiently as possible. The approach that is supposed to answer these questions starts by building an MT application based on the concept of Service-oriented Architecture (SOA). This means the so-called composite SaaS-Applications are composed of a number of application components (AC) that each offer atomic functionality. The approach enables the customer to choose if or even with whom they want to share a specific AC. This way customers may declare their requirements toward a deployment of their application variant.
As an example please refer to figure 1. It illustrates a setting in which six tenants are using an application that is composed of three ACs. Example requirements that would be possible are:

1) AC 1 is not handling any critical data, thus all tenants feel comfortable to share instances.
2) AC 2, however, handles semi-sensitive data. Thus,

978-0-7695-5028-2/13 $26.00 © 2013 IEEE
DOI 10.1109/CLOUD.2013.42

tenant 2, for example, only wants to share an instance with European companies but not with competitors.
3) AC 3 is handling very sensitive data. Thus, tenants will require quite restrictive deployments. Tenant 2, for example, stated that they do not wish to share an instance with competitors and only with companies from the USA.
4) Tenant 6 is special in the sense that they demand a single instance for their private use of every AC.

The example will be discussed in more detail in Section VI to conclude the paper.
This paper's contribution is a semantic model that allows customers to describe their constraints towards a mixed-tenancy deployment. Furthermore, this paper will demonstrate how the information of which tenants may actually share a specific AC can be extracted from that kind of model. In order to come up with such a model, right after a discussion about related work, the paper starts by analyzing requirements customers may have and illustrating how such a model may be used. Based on these discussions a realization of the model is introduced. At the end the entire model will be applied to an example in order to demonstrate its capabilities.

II. Related Work

SaaS in general and the configuration issues and challenges related to it were explored in [3]. The aspect of functional variability has been discussed in some work. As, for example, in [1] the author discusses bringing such flexibility to the process layer of process-based, service-oriented SaaS-Applications. This is done by creating variability descriptors that are transformed into a BPEL process model. In [4] an approach is presented to describe variability for SaaS-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 functional variability. Furthermore, there are approaches with the goal to realize MT that meets privacy and performance requirements. In [5], for example, a framework for multi-tenant aware SaaS-Applications including data isolation, performance isolation or configuration is described. Our approach in contrast allows the customer to give input for how and with whom their ACs are deployed. Data security is achieved as tenants do not share the same AC instances. To the best of our knowledge there is no similar approach. A first plump attempt has been published in [6]. In addition, approaches that realize tenants' isolation on an MT instance may be applied.

III. Analysis of Requirements towards Mixed-tenancy Description

In order to come up with a model to capture the constraints customers may have towards their mixed-tenancy deployment, it is necessary to start by analyzing what the model is supposed to be able to capture.

A. Levels of Deployment

The first step is to analyze what the model is going to be used for, once it has been created. In order to do that it is worthwhile to take a look at what a deployment has to look like. Since this is what shall be created based on the requirements expressed by the model.
The goal of this work is to deploy composite SaaS-Applications following the mixed-tenancy paradigm. MT refers to have multiple tenants sharing one instance of an application and the underlying infrastructure stack. A single-tenancy application, on the other hand, is only used by one tenant. It may, however, be possible that the Service Provider chooses to have tenants share some of the infrastructure stack (e.g. shared virtual infrastructure).
It is the goal of this work to allow customers to express their deployment constraints for different layers of the infrastructure stack. These are referred to as Deployment Levels (DL). A DL is an infrastructure layer for which the Service Provider wishes to realize mixed-tenancy. There may be one to many chosen by the Service Provider. For each, customers may express their deployment constraints. Similar to the infrastructure stack, the DLs are structured as a stack, each having none or one predecessor (lower) and none or one successor (next higher) DL. An example DL-Stack may be AC, Application Server, Virtual Machine, Virtual Infrastructure. If an infrastructure layer may become a DL depends on whether the following requirements are met:

1) A Deployment Level for ACs needs to be defined: This work is dedicated to composite SaaS-Applications. Thus, its requirement is mandatory to realize the mixed-tenancy approach.

2) It is possible to create multiple instances of the Deployment Level: This is mandatory since the basic idea of this work is to increase privacy by separating tenants.

3) The predecessor (lower) Deployment Level shall be able to run multiple instances of the current Deployment Level simultaneously: The requirement is necessary, as otherwise it would not be possible for the tenants to express their constraints for every DL independently. An example where this requirement cannot be met is *Operating System* over a *Virtual Machine*. This is due to 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.

4) Sharing of data between different Instances of the same Deployment Level shall be establishable, if necessary: In order to have a working application it may be necessary that the instances of a particular DL may share data. An obvious example for this is creating a DL 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. The Application Server Instances that belong to the same tenant (and thereby to the user) will need to share the session data in order to function properly.
However, there may be DLs where a communication may not be necessary (e.g. Virtual Machine or stateless ACs). In this case this requirement does not need to be considered.

B. Deployment Models

For every AC and Deployment Level (DL) the customers may choose the Deployment Model in which they want it to run in. A *Deployment Model* may be defined as a blueprint of a deployment. It defines the characteristics of if or how tenants will be able to express their deployment constraints. As discussed in the previous Section, it is necessary for customers to make that choice multiple times, since it needs to be defined for every DL. The same Deployment Model may be applied to multiple (or even all) ACs and/or multiple (or even all) levels of deployment. This means it is possible to specify that the entire application is shared freely with other tenants, on all DL. The following are the Deployment Models that shall be supported by the model.

1) Private: Following this deployment model, an instance is deployed specifically for a single tenant (no sharing). If applied to the AC Deployment Level the entire application stops being a pure MT application since this deployed AC does not need to support MT anymore. In fact, it would be possible that a customer demands that all ACs are deployed following the private model. This would lead to a single-tenancy deployment of an MT application. If another model would be chosen for lower Deployment Levels, it would still be possible for the Service Provider to do some resource usage optimization.

2) Public: This deployment model allows a free number of tenants to share the same instance. The tenants have no influence on with whom they are deployed. As this is the easiest way for the operator to deploy the application, it is supposed to be the cheapest for the customer.

3) White Hybrid: This deployment model allows customers to specify with which specific tenants or groups of tenants they feel comfortable to share instances. An application of this might be that a company demands that only individual companies, belonging to the same company group share an instance. Furthermore, this deployment model allows to manage collaboration if applied to the AC Deployment Level. Since it can be used to specify which specific customers want to share the same AC instance that allows collaboration, for example by deactivating isolation of an MT component.

4) Black Hybrid: In this deployment model customers can specify with which other tenants they do not want to be deployed. A real customer demand that can be fulfilled using this model is that a customer demands not to be deployed with competitors (without specifically specifying who they are).

5) Gray Hybrid: This deployment model allows customers to do both, specify with whom they want to be deployed and with whom they do not wish to be deployed. This model allows customers the most freedom to describe their constraints towards the deployment. Thus, the model allows customers to specify, for example, that they want to share an instance only with tenants from a specific geographic region but not their competitors.

C. Groups

Some of the previously defined Deployment Models provide the ability to the customer to explicitly specify with which other tenants they wish or do not wish to share. This inclusion and exclusion, however, shall not only be done based on individual tenants. In fact, it shall be possible to exclude or include entire groups of tenants. A *Group* may be defined as an entity to which a collection of tenants may be associated. Each *Group* represents a communality that all associated tenants share. For example, it would be possible to create a group that represents a particular industry (e.g. IT, TC, Pharmaceutical). This group may then be used to associate all tenants that belong to this particular industry. Thus, it is possible to realize the requirement that a tenant wishes not to share infrastructure with competitors, by excluding the particular industries this tenant operates in.

D. Dimensions

The previous Section grouped tenants according to the industries they operate in. This industry-based grouping is only one possible way of grouping tenants. In fact there are a lot of others possible. The ways of grouping tenants, we refer to as *Dimensions*. A Dimension is defined as a collection of commonalities that contribute to the same subject. It is realized by a collection of *Groups*. There shall be one to many dimensions possible. Since they may be different depending on the application, this work is not limited to any particular Dimension. In fact, we assume that the Service Providers decide which dimensions are needed. It shall be possible to organize the groups realizing the same dimension as tree-structure (where the root is the dimension). The lower right part of Figure 2 illustrates this structure. As visualized by the figure it is possible that groups have sub-groups that are a sub-set of them. The advantage of structuring

groups like that is that it is possible to gain a much finer-grained categorizing 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). Once customers express to include or exclude a particular group they will also include or exclude all groups that are a sub-group of them as well as all tenants associated (transitive including groups/tenants).

E. Virtual Tenants

It was just stated that tenants shall be able to explicitly include or exclude specific tenants. This may be a problem as a tenant might want to exclude companies that are not tenants yet. The idea of this work is to tackle this problem with the goal of keeping customer base secret. This may be accomplished by introducing the concept of virtual tenants. A virtual tenant is a company that is not yet a customer of the application. Once a company becomes a tenant of the application, it may be transformed to a regular tenant by keeping all previously specified constraints valid.

IV. Process of Customers' Constraint definition

The previous Section discussed the requirements tenants may have towards the description of mixed-tenancy deployments. This Section, however, starts to present the solution approach of the description problem. It discusses the application of the model by describing the process of how the constraints are defined. This is done by discussing the necessary steps that belong to the process.

A. Customizing

It was an objective of this work to create a model that 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 model to the specific application that is supposed to be offered. Thus, this step is performed by the Service Provider.

1) Deployment Level Definition: This activity's purpose is the definition of the DLs. For this the requirements stated in Section III-A need to be fulfilled. Furthermore, it was stated that tenants will have the ability to express their deployment constraints for every DL. Thus, Service Providers should only create DLs for those infrastructure layers for which they want to permit constraints for. There is no restriction to the number of DLs that may be created. However, the creation of DLs has impact on how optimal available resources may be utilized.

2) Dimension and Group Definition: Based on Dimensions and Groups tenants may exclude or include other tenants from the sharing of resources (was discussed in Section III-C and III-D). The goal of this activity is the definition of those dimensions and the groups that realize them. For this the model only provides 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 Service Provider to determine the specific Dimensions they want to provide to their customers. This allows the Service Provider the chance to take the requirements of their customers into account and define dimensions according to their needs. Once the Dimensions have been defined, creation of groups needs to be done based on the target customer base. Since groups will represent the commonalities that customers may have, the target customer base will give some insight on which groups may be necessary (e.g. industries in which the customers are expected to operate in). The level of abstraction groups represent, however, needs to be determined by the Service Provider. 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.
How the Service Provider 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 constraints the less restrictive will their deployments 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 Service Provider only creates a finance group, the customer will exclude a higher number of tenants, which leaves the Service Provider with less space for optimization. Thus, it is in the interest of the Service Provider to have very detailed means of categorizing tenants within dimensions.

B. Tenant Grouping

The second step of the process is called tenant grouping. Similar to the previous step it is also executed by the Service Provider. 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 per dimension (may be done transitively). This is due to the fact that otherwise it would not be certain that excluding a particular group may actually result in excluding all tenants with this commonality.
This process step appears easy at first glance, however, it is not, once looked at closer. The tenant grouping is a crucial step in order to create data on which constraints may be defined. If this data is flawed, 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 (neither directly nor transitively), so an exclusion of the Asian group will not lead to a deployment without this particular customer. Such problems (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), two companies may merge, or

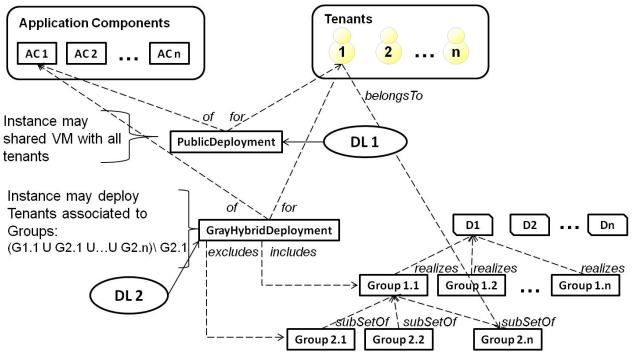

Figure 2. Description of the Requirement

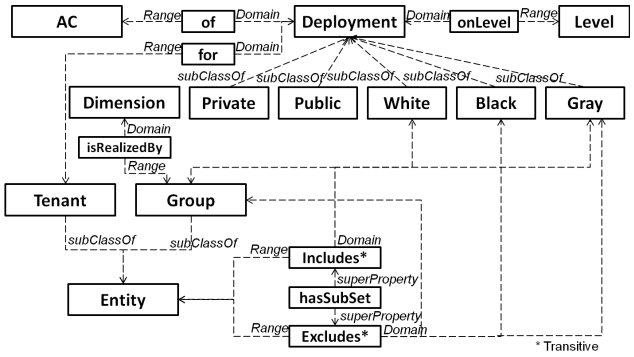

Figure 3. OWL-Model that allows to describe customers' deployment constraints (figure does not capture isCustomer data property)

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 it is crucial that validation processes and techniques are established that guarantee data validity. This is out of scope for this work as it is limited to the techniques on how these requirements may be described.

C. Constraint Definition

It is the purpose of this Section to put previous discussions about Dimensions, Groups, Grouping, and Deployment Levels together and discusses how customers may express their constraints for a deployment of a particular component.
Figure 2 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 AC (or ACs) and to which Deployment Level(s) the deployment shall apply. Based on this the final step is to assign 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 to specifically specify which groups or tenants shall be excluded and which shall be included. Figure 2 indicates this by the deployment called "GrayHybridDeployment", which is for tenant 1, deploying AC 1 and applying to DL 1. In this example tenant 1 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 1 only wants to share an instance with tenants associated to the groups 1.1, 2.2, . . . , 2.n.
As discussed in Section III-A it may be necessary that a tenant specifies the deployment of an AC multiple times. This is why in the example of fig. 2 tenant 1 specifies two deployments of AC 1, one which is on deployment level 1 and one which is on deployment level 2.

D. Deployment Information Extraction

Once all tenants have expressed their constraints for all ACs they use on all levels, it is possible to extract the information that is needed for calculating valid deployments. The information needed to do so is whether two tenants are allowed to share infrastructure or not. This is only the case if neither one of them has constraints that express that they shall not share resources. This information may easily be represented per AC and DL by an undirected graph (nodes represent tenants and edges represent whether two tenants may share infrastructure or not). Such a graph is needed for every AC, once per Deployment Level (Number of graphs = Number of Deployment Levels $\times$ Number of ACs). All those graphs shall automatically be created once all constraints have been described by the tenants. Since they represent an input for the algorithm calculating a consistent and optimal algorithm.

V. Realization using Semantic Web Technologies

In the previous Sections the requirements and the process to describe customer constraints towards mixed-tenancy have been analyzed.
This Section's purpose is to demonstrate how the model may be implemented using Semantic Web Technologies. The advantage of using these technologies lies in the structure of the groups and the transitive exclusion and inclusion. This may very easily be described using Semantic Web Technologies. This is discussed in the first part of this Section. Furthermore, these technologies provide a very easy and straight forward way to extract the deployment information.

A. Introduction of the OWL Model

The OWL-model illustrated by figure 3 was created based on the requirements that were analyzed in previous Sections. It consists of the classes *AC*, *Tenant*, *Level*, *Group*, and

Dimension which all represent the entities that were discussed in previous Sections. In addition, there is a property called *isRealizedBy* between *Dimension* and *Group*. It is used to associate groups to the dimensions they belong to. Another important element is the class called *Deployment*. This class is the domain of the three properties *of*, *for*, and *onLevel*. These three properties serve to determine which tenant specifies the *deployment*, which *ACs* are in the *deployment*, and to which Deployment *Level* it shall apply to. Furthermore, *Deployment* has five sub-classes (*Private*, *Public*, *White*, *Black*, and *Gray*). Each of those represents a particular deployment model. This expresses that once an object of *Gray* is created it is also of type *Deployment*.

In order to capture virtual tenants (discussed in Section III-E) the isCustomer (Domain: Tenant) data property is used. It allows the creation of virtual tenants by stetting it to false.

In Section IV-A2 it was discussed that it shall be possible to structure groups as directed graphs without cycles. This is realized in this model by applying the composite pattern [7]. First, there is a super-class for *Tenant* and *Group* called *Entity*. In Section IV-C it was defined that any time a group is excluded or included all groups that are sub-sets to this group shall be excluded or included as well. Thus, both properties are defined transitively and, thereby, will include and exclude down to the leaves of the tree. Further, both properties' ranges are defined as *Entity*. The domain of property *Includes* is defined as: "*Group* or *GrayHybrid* or *WhiteHybrid*", and for property '*Excludes*: "*Group* or *GrayHybrid* or *BlackHybrid*". This is due to the fact that the White Hybrid deployment model may include tenants, the Black Hybrid model may exclude, and the Gray Hybrid model may do both. Both properties are super-properties to the property *hasSubSet*. Using this property it is possible to create the desired structure of groups and it is possible to assign tenants to groups.

Based on the model it is possible for the OWL-Reasoner to exclude or include all groups according to the structure defined by the service provider. This is going to be demonstrated in Section VI where the model is applied to the example.

B. Extraction of Graphs

The information necessary for the next step is whether two tenants are allowed to share infrastructure, i.e. the same application component at the same level. A feasible representation for this information is an undirected graph where nodes represent tenants and edges represent that they may share infrastructure, as has been discussed in Section IV-D. The information needs to be extracted from the OWL-based model with the help of a two-step process. In the first step the user-friendly specifications of all requirements are completed by relating tenants (and not only groups) to deployment models 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.

1) Step 1: Introduction of "mayDeployTenant"-Property: The first step toward extraction of the deployment graph is to introduce one additional property called "mayDeployTenant" to the model. Its purpose is to associate the tenants that may be deployed together directly to every deployment . This additional property is not simply the closure of the transitive property "Includes" but also requires to remove those tenants/groups that are excluded - depending on the deployment model. Therefore, the following rules are described in an SWRL-like notation how "mayDeployTenant" is introduced into the OWL model:

- **Assigning the Owner** Any deployment may always deploy the owner. This is expressed by: Deployment(?d) $\wedge$ Tenant(?t) $\wedge$ for(?d,?t) $\Rightarrow$ mayDeployTenant(?d, ?t)
- **Private** A private deployment may only deploy the owner. This has been taken care of.
- **Public** A public deployment may deploy any tenant that is using the application, thus[1]: 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) $\Rightarrow$ mayDeployTenant(?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 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) can not 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 implemented 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).

2) Step 2: SPARQL-Queries to extract graph : Once the "mayDeployTenant"-Property has been inserted in the model, the desired deployment information of a given application component `*AC*` and level `*L*` may be extracted by using the following SPARQL-Query.

[1]This rule is not a mistake. Because of the definition of a public deployment model, *any* tenant can be deployed to *any* public deployment.

```
PREFIX ns: < nameSpace >
SELECT ?t1 ?t2 WHERE {
   ?t1 a ns:Tenant .
   ?t1 ns:isCustomer true .
   ?t2 a ns:Tenant .
   ?t2 ns:isCustomer true .
   ?deplyOfAC1 ns:deploymentOf *AC* .
   ?deplyOfAC2 ns:deploymentOf *AC* .
   ?deplyAC1 ns:deploymentFor ?t1 .
   ?deplyOfAC2 ns:deploymentFor ?t2 .
   ?deplyOfAC1 ns:mayDeployTenant ?t2 .
   ?deplyOfAC2 ns:mayDeployTenant ?t1 .
   ?deplyOfAC1 ns:onLevel *L* .
   ?deplyOfAC2 ns:onLevel *L*
}
```

The query returns a list of two tenants per line.

```
--------------------
| 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. This line is only gained if both tenants have agreed to share infrastructure. In order to be safe that even tenants, that have no connections to other tenants are gained, every tenant will appear at least once, associated to himself (A $\rightarrow$ A).

VI. Example Application

This chapter's purpose is to discuss the presented model by applying it to an example. In fact, the example has briefly been introduced in Section I. This chapter, however, will discuss it in more detail. This example allows to demonstrate all capabilities of the model and presents the extracted deployment information.

A. Dimensions

As discussed in Section IV the first step when using the proposed model is that the Service Provider defines the dimensions based on which tenants are going to be grouped. For the example two dimensions were defined: Industry (in which tenants do business), and Geographic (in which country do the companies have their head quarter). These are depicted by figure 4.

B. Tenants and Grouping

The example assumes that there is a total of six tenants. Each of the tenants has been grouped according to the two dimensions. This is illustrated by the following table.

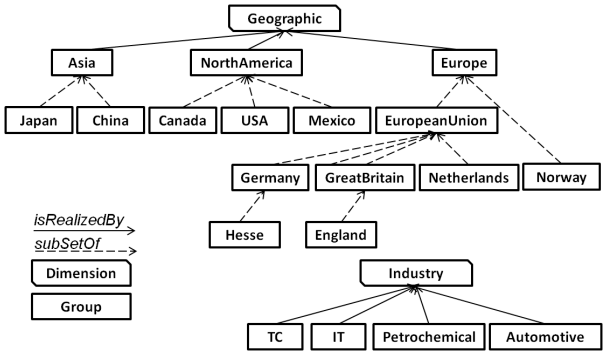

Figure 4. Description of the Example Dimensions and Groups

Tenant	Industry	Geographic
T-1	IT	USA
T-2	Automotive	USA
T-3	Automotive	Hesse
T-4	Petrochemical	Netherlands, England
T-5	Automotive	Japan
T-6	TC, IT	Germany

Each tenant has been associated to at least one group per dimension. However, more are possible (e.g. T-4 has headquarters in England as well as in the Netherlands).

C. Customer Constraints

Before the customer constraints may be described it is first necessary to introduce which ACs are supposed to be described. This is done in the following:

- **AC1** is a component that handles insensitive data. Thus, tenants will not use very restrictive deployments.
- **AC2** handles data of which customers think differently. Thus, here will be a mixture of different deployments used.
- **AC3** handles very sensitive data. Thus, most tenants will desire very restrictive deployments.

Based on those ACs it is possible to capture customers' constraints. These have been captured using the OWL-Model. The following table 5 illustrates all constraints tenants defined for the three available ACs. Please notice that T-4 specifies that they do not wish to share an Instance of AC2 with T-9, even though T-9 is not a customer. This constraint is valid since T-9 was created as a virtual tenant (Section III-E).

D. Graph Extraction

Section V-B discussed how deployment information may be extracted from the OWL-model. Figure 6 gives an overview of the results. The figure represents the deployment information as graphs (node = tenant, edge = may be deployed together). It does that for each AC and level.
The following are some points that may be seen in the resulting graphs:

		AC 1	AC 2	AC 3
T-1	Instance	Public		
	VM			
T-2	Instance	Public		Gray: excl{Automotive} incl{USA}
	VM	Public		
T-3	Instance	Public	Black: excl{Automotive}	Gray: incl{Europe} excl{Automotive}
	VM	Public		Black: excl{Asia}
T-4	Instance	Black excl{Petrochemical, T9}	Gray incl{EUDP, USDP, Europe) excl{T9}	Gray: incl{Europe} excl{T9}
	VM	Public	White incl{Europe}	
T-5	Instance	Black: excl{NorthAmerica, Europe, China}	Private	
	VM	Public	Gray: incl{Europe, Japan} excl{Automotive}	Gray: incl{Japan } excl{Automotive}
T-6	Instance	Private		
	VM			

Figure 5. Table displaying customers' constraints towards the deployment

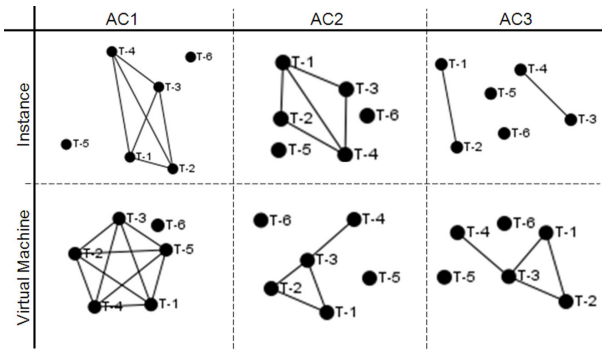

Figure 6. Visualization of the extracted Deployment Information

1) T-1: specified that they wish to use the public deployment for all ACs on both levels. In the deployment information (figure 6) they are being represented as nodes with edges attached. Even though T1 agreed to share resources with everyone, they are not sharing an edge with every other tenant. This is due to the fact that other tenants may have expressed a constraint that states that they do not wish to share resources with T1.

2) T-6: specified that they wish not to share instances or VMs with other tenants. Thus, in the deployment information (figure 6) they appear as nodes without edges attached in every diagram. This means that they may not share resources.

VII. Summary and Conclusion

The objective of this paper was to present a description approach that allows to capture customers' constraints towards the deployment of mixed-tenancy applications. In this model the first step that needs to be done is that the service provider offering the application needs to define dimensions according to which tenants are grouped by (e.g. Industries, Data Protection Laws, or Geographic). Dimensions may be realized by a number of groups, which may be structured as directed graphs without cycles. Once dimensions are defined all tenants need to be associated to at least one group per dimension. Based on this customers express their constraints per AC and per level of deployment (Instance or Virtual Machine) based on the five deployment models: Private, Public, White Hybrid, Black Hybrid, and Gray Hybrid. To be more precise, depending on the chosen deployment model tenants may have the ability to exclude or include specific tenants or groups of tenants. This allows tenants to express, for example, that they wish to share a particular component only with tenants from Europe but not with their competitors (company operating in the same industry as they do).
Furthermore, the paper showed how the model may easily be realized by utilizing technologies from the area of Semantic Web. This includes the way how the pure deployment information was extracted. The capabilities of the model were demonstrated by applying it to an example.
Based on the presented approach the next step is to define an algorithm that finds the most efficient deployment of the application that satisfies the operators' needs.

References

[1] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications," in *Workshop on Principles of Engineering Service Oriented Systems, ICSE*. IEEE Computer Society, 2009, pp. 18–25.

[2] Cloud Security Alliance, "The notorious nine: Cloud computing top threats in 2013," Tech. Rep., Feb. 2013. [Online]. Available: https://cloudsecurityalliance.org/download/the-notorious-nine-cloud-computing-top-threats-in-2013/

[3] 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*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 18–25.

[4] 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*, Dec. 2010, pp. 255–260.

[5] 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)*, Jul. 2007, pp. 551–558.

[6] S. T. Ruehl, U. Andelfinger, A. Rausch, and S. A. Verclas, "Toward realization of deployment variability for software-as-a-service applications," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 622 –629.

[7] E. Gamma, R. Helm, and R. E. Johnson, *Design Patterns. Elements of Reusable Object-Oriented Software.*, 1st ed. Addison-Wesley Longman, Amsterdam, Oct. 1994.