Delft University of Technology
Software Engineering Research Group
Technical Report Series

Challenges of Reengineering into Multi-Tenant SaaS Applications

Cor-Paul Bezemer, Andy Zaidman

Report TUD-SERG-2010-012

TUD-SERG-2010-012

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Paper submitted to the 1st Workshop on Engineering SOA and the Web (ESW'10).

Challenges of Reengineering into Multi-Tenant SaaS Applications

Cor-Paul Bezemer[1,2] and Andy Zaidman[1]

[1] Delft University of Technology, the Netherlands
[2] Exact, Poortweg 6, 2612 PA Delft, the Netherlands, http://www.exact.com
{c.bezemer, a.e.zaidman}@tudelft.nl

Abstract. Multi-tenancy is a relatively new software architecture principle in the realm of the Software as a Service (SaaS) business model. It allows to make full use of the economy of scale, as multiple customers – "tenants" – share the same application and database instance. All the while, the tenants enjoy a highly configurable application, making it appear that the application is deployed on a dedicated server. The major benefits of multi-tenancy are increased utilization of hardware resources and improved ease of maintenance, resulting in lower overall application costs, making the technology attractive for service providers targeting small and medium enterprises (SME). In our paper, we identify some of the core challenges of implementing multi-tenancy. Furthermore, we present a conceptual reengineering approach to support the migration of single-tenant applications into multi-tenant applications.

1 Introduction

Software as a Service (SaaS) represents a novel paradigm and business model expressing the fact that companies do not have to purchase and maintain their own ICT infrastructure, but instead, acquire the services embodied by software from a third party. The customers subscribe to the software and underlying ICT infrastructure (service on-demand) and require only Internet access to use the services. The service provider offers the software service and maintains the application [8]. However, in order for the service provider to make full use of the economy of scale, the service should be hosted following a multi-tenant model [10].

Multi-tenancy is an architectural pattern in which a single instance of the software is run on the service provider's infrastructure, and multiple tenants access the same instance. In contrast to the multi-user model, multi-tenancy requires customizing the single instance according to the multi-faceted requirements of many tenants [10]. The multi-tenant model also contrasts the multi-instance model, in which each tenant gets his own instance of the application [2].

The benefits of the multi-tenant model are twofold. On one hand, application maintenance becomes easier for the service provider, as only one application instance has to be maintained. On the other hand, the utilization rate of the hardware can be improved. These two factors reduce the overall costs of the application and this makes multi-tenant applications especially interesting for

customers in the small and medium enterprise (SME) segment of the market, as they often have limited financial resources and do not need the computational power of a dedicated server.

Because of these benefits, many organizations working with SaaS technology are currently looking into transforming their single-tenant applications into multi-tenant ones. Yet, they see a major barrier in the reengineering process that they should go through for this transformation [15]. This is where this paper aims to contribute, by specifically addressing the challenges and difficulties of this process. More specifically, our paper contains the following contributions:

1. A clear, non-ambiguous definition of a multi-tenant application.
2. An overview of the challenges of developing and maintaining scalable, multi-tenant software.
3. A description of the requirements of reengineering a single-tenant into a multi-tenant application.
4. A conceptual reengineering approach for supporting this process.
5. A case study of reengineering an open source single-tenant application into a multi-tenant application.

This paper is further organized as follows. In the next section, we give a definition of multi-tenancy and discuss its benefits and related work. In Section 3, we discuss the challenges of multi-tenancy. In Section 4, we present our reengineering approach for the transformation of a single-tenant to a multi-tenant application. Section 5 covers our case study of reengineering a single-tenant open source application into a multi-tenant application. We conclude our paper with a discussion and concluding remarks.

2 Multi-Tenancy

Multi-tenancy is an organizational approach for SaaS applications. Although SaaS is a business model, its introduction has lead to numerous interesting problems and research in the web application community. Despite this research, we believe multi-tenancy has not yet received the attention it deserves. A number of definitions of a multi-tenant application exist [17, 18], but we believe these are all unclear. Therefore, we define a multi-tenant application as the following:

Definition 1. *A **multi-tenant** application lets customers (**tenants**) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment.*

Definition 2. *A **tenant** is the organizational entity which rents a multi-tenant SaaS solution. Typically, a tenant groups a number of users, which are the stakeholders in the organization.*

These definitions focus on what we believe to be the key aspects of multi-tenancy:

1. Hardware resource sharing.
2. High degree of configurability.
3. Shared application and database instance.

In the next paragraphs we will explain the difference between a multi-user and a multi-tenant application and elaborate on the key aspects of multi-tenancy.

2.1 Multi-Tenant versus Multi-User

It is necessary to make an important, but subtle distinction between the concepts *multi-tenant* and *multi-user*. In a multi-user application we assume all users are using the same application with limited configuration options. In a multi-tenant application, we assume each tenant has the possibility to configure the application heavily. This results in the situation that, although tenants are using the same building blocks in their configuration, the appearance or workflow of the application may be completely different for two tenants. An additional argument for the distinction is that the Service Level Agreement (SLA) of each tenant can differ, while this is usually not the case for users in a multi-user system.

2.2 Hardware Resource Sharing

In traditional single-tenant software development, tenants usually have their own (virtual) server. Usually, in the SME segment, utilization of such a server is low. By placing several tenants on the same server, the server utilization can be improved [16, 17]. Although this happens when virtualization is used as well, virtualization imposes a much lower limit on the number of tenants per server due to the relatively high memory requirements for every virtual server [12]. Higher utilization of the existing servers will result in lower overall costs of the application, as the total amount of hardware required is lower. The concept of multi-tenancy comes in different flavours, and depending on which flavour is implemented, the utilization rate of the underlying hardware can be maximized. The following variants of (semi-)multi-tenancy can be distinguished [2, 10]:

1. Shared application, separate database
2. Shared application, shared database, separate table
3. Shared application, shared table (**pure multi-tenancy**)

Throughout this paper, we will assume the pure multi-tenancy variant is being used, as the other two have performance issues when a large number of tenants are placed on the same server [2, 16].

2.3 High degree of configurability

In a single-tenant environment, every tenant has his own, (possibly) customized application instance. In multi-tenancy, all tenants share the same application instance, although it must appear to them as if they are using a dedicated one. Because of this, a key requirement of multi-tenant applications is the possibility to configure and/or customize the application to a tenant's need. In single-tenant software customization is often done by creating branches in the development tree. In multi-tenancy this is no longer possible and configuration options must be integrated in the product design instead [14]. Another feature that is essential to the design of multi-tenancy is version support, as it may be necessary to run multiple versions of an application (e.g., for backwards compatibility). Because it is undesirable to deploy different instances of a multi-tenant application, version support must be integrated as a configuration feature.

2.4 Shared Application and Database Instance

A single-tenant application may have many running instances and they may all be different from each other because of customization. In multi-tenancy, these differences no longer exist as the application can be configured at runtime.

The overall number of instances will clearly be much lower (ideally it will be one, but the application may be replicated for scalability purposes). As a consequence, maintenance is much easier and cheaper, as updates have to be unrolled on a small number of instances only. In addition, new data aggregation opportunities are opened because all tenant data is in the same place.

2.5 Benefits

From the previous paragraphs a number of reasons for companies to change their software to be multi-tenant can be deducted:

1. Higher utilization of hardware resources. (§2.2)
2. Easier and cheaper application maintenance. (§2.4)
3. Lower overall costs, resulting in the opportunity to offer a service at a lower price than competitors. (§2.2, §2.4)
4. New data aggregation opportunities. (§2.4)

2.6 Related Work

Even though SaaS is an extensively researched topic, multi-tenancy has not received a large deal of attention yet in academic software engineering research. A number of researchers [2, 5, 10] have described the possible variants of multi-tenancy, as we have described in Section 2.2. Wang et al. [16] have evaluated these variants for different numbers of tenants and make recommendations on the best multi-tenant variant to use, based on the number of tenants, the number of users per tenant and the amount of data per tenant.

Kwok et al. [10] have described a case study of developing a multi-tenant application, in which they emphasize the importance of configurability. This importance is emphasized by Nitu [14] and Mietzner et al. [13] as well, although the latter approach still requires deploying a different module for every tenant.

Guo et al. [5] have proposed a framework for multi-tenant application development and management. They believe the main challenge of multi-tenancy is tenant isolation, and therefore their framework contains mainly components for tenant isolation, e.g., data, performance and security isolation. We believe tenant isolation forms a relatively small part of the challenges of multi-tenancy, which is why our paper focuses on different aspects.

The native support of current database management systems (DBMSs) for multi-tenancy was researched by Jacobs and Aulback [7]. In their position paper on multi-tenant capable DBMSs, they conclude that existing DBMSs are not capable of natively dealing with multi-tenancy. Chong et al. [2] have described a number of possible database patterns, which support the implementation of multi-tenancy, specifically for Microsoft SQL Server.

One problem in multi-tenant data management is tenant placement. Kwok et al. [11] have developed a method for selecting the best database in which a

new tenant should be placed, while keeping the remaining database space as flexible as possible for placing other new tenants.

Finally, Salesforce, an industrial pioneer of multi-tenancy, has given an insight on how multi-tenancy is being handled in their application framework [18].

3 Challenges

Unfortunately, multi-tenancy has its challenges and even though these challenges exist for single-tenant software as well, they appear in a different form and are more complex to solve for multi-tenant applications. In this section we will list the challenges and discuss their specificity with regard to multi-tenancy.

3.1 Performance

Because multiple tenants share the same resources and hardware utilization is higher on average, we must make sure that all tenants can consume these resources as required. If one tenant clogs up resources, the performance of all other tenants may be compromised. This is different from the single-tenant situation, in which the behaviour of a tenant only affects himself. In a virtualized-instances situation this problem is solved by assigning an equal amount of resources to each instance (or tenant) [12]. This solution may lead to very inefficient utilization of resources and is therefore undesirable in a pure multi-tenant system.

3.2 Scalability

Because all tenants share the same application and datastore, scalability is more of an issue than in single-tenant applications. We assume a tenant does not require more than one application and database server, which is usually the case in the SME segment. In the multi-tenant situation this assumption cannot help us, as such a limitation does not exist when placing multiple tenants on one server. In addition, tenants from a wide variety of countries may use an application, which can have impact on scalability requirements. Each country may have its own legislation on, e.g., data placement or routing. An example is the European Union's (EU) legislation on the storage of electronic invoicing, which states that electronic invoices sent from within the EU must be stored within the EU as well[3]. Finally, there may be more constraints such as the requirement to place all data for one tenant on the same server to speed up regularly used database queries. Such constraints strongly influence the way in which an application and its datastore can be scaled.

3.3 Security

Although the level of security should be high in a single-tenant environment, the risk of, e.g., data stealing is relatively small. In a multi-tenant environment, a security breach can result in the exposure of data to other, possibly competitive, tenants. This makes security issues such as data protection [5] very important.

[3] `http://ec.europa.eu/taxation_customs/taxation/vat/traders/invoicing_rules/article_1733_en.htm` (last visited on Feb 09, 2010)

3.4 Zero-Downtime

Introducing new tenants or adapting to changing business requirements of existing tenants brings along the need for constant growth and evolution of a multi-tenant system. However, adaptations should not interfere with the services provided to the other existing tenants. This induces the strong requirement of zero-downtime for multi-tenant software, as downtime per hour can go up to $4,500K depending on the type of business [4].

4 Architectural Approach for Multi-Tenancy

In order to be able to address the challenges discussed in Section 3, the architecture of the traditional three-tier single-tenant web application [6] must be adapted. We propose an architectural approach for reengineering single-tenant to multi-tenant applications, shown in Figure 1. Our approach is suitable for making this transformation, as the components can be integrated by making minor changes to the original, single-tenant source code. In this section we will elaborate on the purpose of each layer shown in Figure 1.

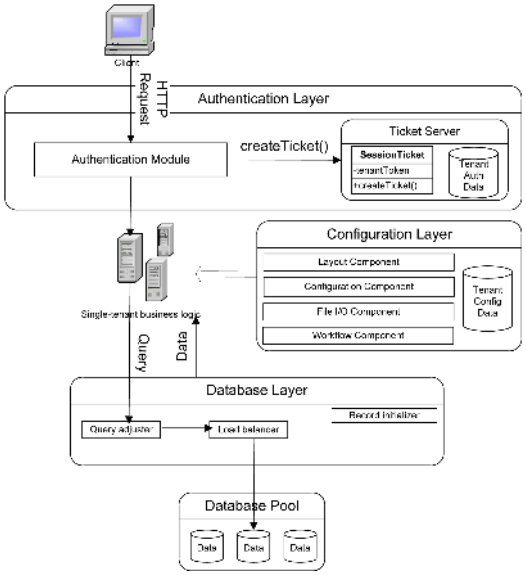

Fig. 1. Multi-tenant reengineering approach

4.1 Authentication

The purpose of the authentication layer is to provide a mechanism for identifying each tenant in the application, by generating a tenant ticket (token) after a tenant successfully logs in. This ticket can be used throughout the application to load the corresponding configuration for a tenant. To provide a straightforward and efficient authentication verification mechanism, a single-sign on protocol, such as Kerberos [9] can be used for ticket generation. The advantage of such

a protocol is that all multi-tenant components can be implemented as stand-alone services, which means they are better scalable. In addition, there is no need to contact the authentication server as all components can easily verify the authenticity of a tenant using the ticket.

4.2 Configuration

In order to make a single-tenant application multi-tenant capable, it is necessary to allow at least the following types of configuration:

Layout The layout configuration component allows the use of tenant-specific themes.

General configuration The general configuration component allows the specification of tenant-specific configuration, such as database settings, encryption key settings and personal profile details.

File I/O The file I/O configuration component allows the specification of tenant-specific file paths, which can be used for, e.g., report generation.

Workflow The workflow configuration component allows the configuration of tenant-specific workflows. An example of an application in which workflow configuration is required is an enterprise resource planning (ERP) application, in which the workflow of requests can vary significantly for different companies.

4.3 Database

Perhaps the most important difference between a single-tenant and multi-tenant application is the greater requirement for data management and isolation. Because current off-the-shelf DBMSs are not capable of dealing with multi-tenancy themselves [7], this should be done in a layer between the business logic and the application's database pool. The main tasks of this layer are as follows:

Creation of new tenants in the database If the application stores and/or retrieves data, which can be made tenant-specific, in/from a database, it is the task of the database layer to create the corresponding database records when a new tenant has signed up for the application.

Query adaptation In order to provide adequate data isolation, the database layer must make sure that all queries are adjusted so that each tenant can only access his own records.

Load balancing To improve the performance of the multi-tenant application, efficient load balancing is required for the database pool. Note that any agreements made in the SLA of a tenant and any constraints imposed by the legislation of the country a tenant is located in must be satisfied. In addition, the application may have requirements on where a tenant's data is being stored, e.g., for report generation. These requirements make it difficult to use existing load balancing algorithms. On the other hand, our expectation is that it is possible to devise more efficient load balancing algorithms using the information we possess about tenants.

8

5 Reengineering single-tenancy into multi-tenancy

Formally evaluating a reengineering approach is very difficult, which is why we use a case study to investigate the suitability of our choice of multi-tenant components [19]. In this case study, we transform a single-tenant application into a multi-tenant version. We report on the challenges that we encountered during this reengineering operation and we touch upon a number of technological considerations.

In particular, we will use the ScrewTurn Wiki[4] system (referred to as STWiki throughout the rest of this paper) as a case study. An example stakeholder of a transformation of the STWiki system is a hosting provider looking for a cheaper way to offer STWiki to its customers. By offering a multi-tenant version instead of a separate instance for every customer, maintenance costs can be kept much lower as only one instance has to be changed at the time of a program update.

5.1 Case Study: ScrewTurn Wiki

STWiki is an open source wiki application with an active developers community, written in C# and based on the ASP.NET 3.5 platform. In this section we will explain which steps were taken to transform STWiki into a multi-tenant version. Figure 2 depicts how each multi-tenant component was implemented in our case study.

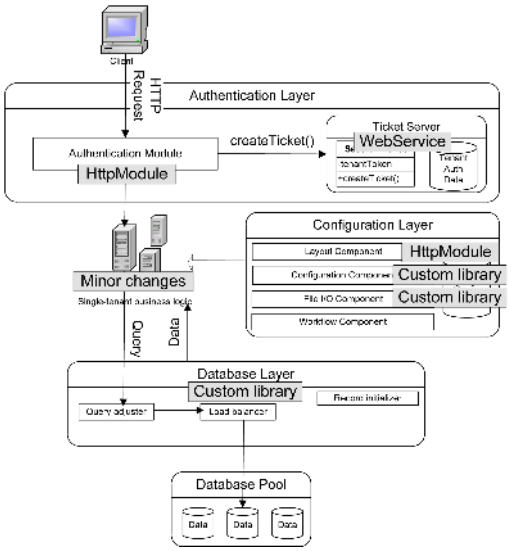

Fig. 2. Implementation of multi-tenant components for ScrewTurn Wiki

[4] http://www.screwturn.eu/

5.2 Authentication

The authentication layer is conceived as two separate components. The ticket generation mechanism is implemented as a web service, which generates tickets using the Kerberos protocol. Such tickets contain enough details to allow tenant identification throughout the application without requiring continuous calls to the Ticket Server web service. In addition, it is possible to add extra information to the ticket when necessary, such as configuration details. Because the ticket is encrypted, a user cannot change this information.

The Authentication Module is implemented as a ASP.NET HttpModule. For every HTTP request, this module verifies whether it comes from an identified tenant. If this is not the case, a login screen is displayed to the tenant, otherwise the STWiki application is shown with the correct configuration. Note that this mechanism does not interfere with the authentication mechanism of STWiki itself, which means that different users represented by the same tenant can still have different roles.

5.3 Configuration

The configuration library consists of three separate components. The layout component is implemented as HttpModule, which loads a tenant-specific master page[5] for each requested page.

The general configuration and file I/O components are implemented as custom libraries and integrated by adapting the STWiki source code to make use of these libraries. Note that only a part of the configuration and file I/O is adapted, as the goal of this case study is to serve as a prototype for our reengineering approach, rather than deliver a fully functioning multi-tenant version of STWiki.

An alternative would be to integrate the libraries using Aspect Oriented Programming (e.g., with PostSharp [3]), but this is future work. Defining file I/O as a crosscutting concern is a tedious task, because of the many different possible ways of doing file I/O in .NET. Configuration is very difficult to define as a crosscutting concern, because no generic configuration mechanism exists. Therefore, it is necessary to manually discover and extend the current configuration mechanism for each application. This may be very easy (loading a tenant-specific XML configuration file) or more difficult (replacing hardcoded configuration values), depending on the existing application.

In our case study we have not reconfigured the workflow of STwiki. However, we have investigated how we could enable this in future case studies and we found that while defining workflow configuration as a crosscutting concern is a difficult task, extension of workflow languages with aspect-oriented programming has been addressed in other research [1].

5.4 Database

The database layer is built as a custom library and integrated by adapting STWiki to use this library. The query adjuster updates the *WHERE* clause of

[5] Master pages are the ASP.NET mechanism for defining page templates.

every SQL query with the statement *tenantId = 'xxx'*, where *xxx* is retrieved from the tenant's session ticket. In this case study, no database load balancing is done.

6 Discussion

Scalability The multi-tenant components defined in our reengineering approach are defined with scalability in mind. They either introduce little computational overhead, such as the file I/O and configuration components, or they may be implemented as stand-alone services on separate servers, such as the session ticket server.

Part of the scalability issues find their origin in database access. Therefore it is important to make database access scalable, e.g., by implementing the load balancer as a stand-alone service such that it can easily be replaced or scaled if necessary. A requirement of the load balancer is that it should allow extension of the database pool by adding new database servers when desired. In addition, placing a load balancer before the Tenant Auth and Config Data databases makes the authentication and configuration components more scalable. It is also possible to move these databases to the main database pool, which may give better results, depending on the load balancing constraints and requirements.

Because we expect the database access to cause the majority of the scalability problems, we have not taken the scalability of the single-tenant business logic into account. However, this may cause problems when more tenants use the system. Further research must be done to find out if multi-tenancy imposes new scalability requirements on the existing single-tenant business logic.

Configurability A multi-tenant application may require much more configurability than the single-tenant business logic currently offers. Even though we recognize configurability as a key aspect of a multi-tenant application, we also realize that the configuration requirements depend heavily on the type and implementation of the application. Therefore, we have decided not to give a detailed specification of configuration requirements.

Version Support In Section 2.3, we mentioned that version support is an essential feature in the design of a multi-tenant application. Supporting different application versions in the same instance is a complex task and will be addressed in future research.

Completeness Specific multi-tenant functionality can be clearly defined as an extension of single-tenancy and we have decided to leave non-specific multi-tenant functionality, such as testing, out of the picture in order to avoid clutter.

Different Applications Even though our reengineering approach is designed with the goal of transforming a single-tenant application into a multi-tenant application, our research is useful for multi-tenant applications created from scratch as well. Our approach emphasizes the key aspects in multi-tenancy, which should be addressed by any multi-tenant application. In fact, we believe that any multi-tenant application should be mappable onto our architectural approach.

Threats to Validity We have discussed some of the issues concerning the external validity of our approach in the above discussion. As far as the internal validity is concerned, we would like to emphasize that a reengineering approach is very difficult to evaluate. To evaluate the correctness of the transformation, we have tested the multi-tenant application by manually verifying a random sampling of functionality.

As multi-tenancy is a relatively new concept, especially in the software engineering world, very little research has been done on this subject. We have defined the multi-tenant components in our approach after having researched existing problems in multi-tenant applications. This research was conducted by analyzing papers, the demand from industrial partners and by reading blog entries (including the comments, which form a source of valuable information as they contain information about the current problems in the SaaS industry).

While we have performed only one case study, the application transformed in our case study is representative because it is open-source and has a large, active developer community. This tends to result in a well-tested code-base. In addition, a wiki is a realistic example of an application, which must be made multi-tenant. In the future, we will perform a case-study on an industrial multi-tenant closed-source application as well.

7 Concluding Remarks

In this paper, we have defined multi-tenancy as a Software-as-a-Service (SaaS) engineering challenge, thereby extending its current classification as a business model. Multi-tenancy is an engineering principle that can (1) lead to improvements in hardware utilization and (2) reduce overall application costs, specifically during the maintenance phase. This in turn leads to a competitive advantage for service providers, in particular those providers targeting the small and medium enterprises (SME) segment. In order to optimize the benefits achieved through implementing multi-tenancy, a number of challenges, mainly in the field of performance, scalability and zero-downtime must be overcome. An additional challenge is reengineering existing single-tenant applications to support multi-tenancy. In this context, we have proposed a reengineering approach, which supports this process. Through a case study with the ScrewTurn wiki system, we have shown that our approach is applicable and executable.

Future work includes more specific research on the challenges of multi-tenancy, in particular performance, scalability and version support in large-scale multi-tenant systems. In addition, we also plan to conduct case studies on industrial multi-tenant applications with a large number of real users.

Acknowledgements We would like to thank Exact for funding the Multi-Tenant Systems project (`http://swerl.tudelft.nl/bin/view/Main/MTS`).

References

1. Charfi, A., Mezini, M.: Aspect-oriented workflow languages. In: On the Move to Meaningful Internet Systems Pt I. LNCS, vol. 4275, pp. 183–200. Springer (2006)

2. Chong, F., Carraro, G., Wolter, R.: Multi-tenant data architecture. http://msdn.microsoft.com/en-us/library/aa479086.aspx (June 2006)
3. Fraiteur, G.: Postsharp - bringing aop to .net. http://www.postsharp.org/ (2010)
4. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM Systems Journal 42(1), 5–18 (2003)
5. Guo, C.J., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: A framework for native multi-tenancy application development and management. In: Proc. of the 9th Int. Conf. on E-Commerce Technology (CEC) and the 4th Int. Conf. on Enterprise Computing, E-Commerce, and E-Services (EEE). pp. 551–558. IEEE CS (2007)
6. Hassan, A.E., Holt, R.C.: Architecture recovery of web applications. Proc. of the Int. Conf. on Software Engineering (ICSE), IEEE CS pp. 349–359 (2002)
7. Jacobs, D., Aulbach, S.: Ruminations on multi-tenant databases. In: Datenbanksysteme in Business, Technologie und Web (BTW), 12. Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme (DBIS), Proc. 7.-9. Mrz. LNI, vol. 103, pp. 514–521. GI (2007)
8. Kaplan, J.M.: Saas: Friend or foe? In: Business Communications Review. pp. 48–53 (June 2007), http://www.webtorials.com/abstracts/BCR125.htm
9. Kohl, J., Neuman, C., Muse, A.: The kerberos network authentication service (v5). http://www.faqs.org/rfcs/rfc1510.html (last visited on Feb 08, 2010) (1993)
10. Kwok, T., Nguyen, T., Lam, L.: A software as a service with multi-tenancy support for an electronic contract management application. In: Proc. of the Int. Conf. on Services Computing (SCC). pp. 179–186. IEEE CS (2008)
11. Kwok, T., Mohindra, A.: Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications. In: Proc. Int. Conf. on Service-Oriented Computing (ICSOC). LNCS, vol. 5364, pp. 633–648 (2008)
12. Li, X.H., Liu, T., Li, Y., Chen, Y.: Spin: Service performance isolation infrastructure in multi-tenancy environment. In: Proc. of the 6th Int. Conf. on Service-Oriented Computing (ICSOC). LNCS, vol. 5364, pp. 649–663 (2008)
13. Mietzner, R., Leymann, F., Papazoglou, M.P.: Defining composite configurable saas application packages using sca, variability descriptors and multi-tenancy patterns. In: Third Int. Conf. on Internet and Web Applications and Services (ICIW). pp. 156–161. IEEE CS (2008)
14. Nitu: Configurability in saas (software as a service) applications. In: Proc. of the 2nd annual conf. on India softw. eng. conference (ISEC). pp. 19–26. ACM (2009)
15. Tsai, C.H., Ruan, Y., Sahu, S., Shaikh, A., Shin, K.G.: Virtualization-based techniques for enabling multi-tenant management tools. In: 18th IFIP/IEEE Int. Workshop on Distr. Systems: Operations and Management (DSOM). LNCS, vol. 4785, pp. 171–182. Springer (2007)
16. Wang, Z.H., Guo, C.J., Gao, B., Sun, W., Zhang, Z., An, W.H.: A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In: Proc. of the Int. Conf. on e-Business Engineering (ICEBE). pp. 94–101. IEEE CS (2008)
17. Warfield, B.: Multitenancy can have a 16:1 cost advantage over single-tenant. http://smoothspan.wordpress.com/2007/10/28/multitenancy-can-have-a-161-cost-advantage-over-single-tenant/ (last visited on Jan 30, 2010) (October 2007)
18. Weissman, C.D., Bobrowski, S.: The design of the force.com multitenant internet application development platform. In: Proc. of the 35th SIGMOD int. conf. on Management of data (SIGMOD). pp. 889–896. ACM (2009)
19. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering – An Introduction. Kluwer (2002)

TUD-SERG-2010-012
ISSN 1872-5392