Delft University of Technology
Software Engineering Research Group
Technical Report Series

Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?

Cor-Paul Bezemer, Andy Zaidman

Report TUD-SERG-2010-031

TUD-SERG-2010-031

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: Accepted for publication in the Proceedings of the 4th International Joint ERCIM/IWPSE Symposium on Software Evolution (IWPSE-EVOL), 2010, ACM.

Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?

Position paper

Cor-Paul Bezemer
Delft University of Technology & Exact
The Netherlands
c.bezemer@tudelft.nl

Andy Zaidman
Delft University of Technology
The Netherlands
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, in particular on the deployment side. These benefits should result in lower overall application costs, making the technology attractive for service providers targeting small and medium enterprises (SME). However, as this paper advocates, a wrong architectural choice might entail that multi-tenancy becomes a maintenance nightmare.

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 [7]. 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 [9].

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 [9]. The multi-tenant model also contrasts the multi-instance model, in which each tenant gets his own instance of the application [2].

IWPSE-EVOL'10 September 20-21, 2010 Antwerp, Belgium

The benefits of the multi-tenant model are twofold. On one hand, application deployment becomes easier for the service provider, as only one application instance has to be deployed. 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, two barriers are perceived in the adoption of multi-tenant software systems, namely:

- Companies are wary of the initial start-up costs of reengineering their existing single-tenant software systems into multi-tenant software systems [14].
- Software maintainers are worried that multi-tenancy might introduce additional maintenance problems stemming from the fact that these new systems should be highly configurable, in the process effectively eliminating the perceived maintenance advantage that multi-tenancy offers through the fact that updates only have to be deployed and applied once.

This is where this paper aims to contribute, by providing an overview of challenges and difficulties that software developers and maintainers are likely to face when reengineering and maintaining multi-tenant software applications. 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 conceptual blueprint of a multi-tenant architecture that isolates the multi-tenant concern as much as possible from the base code.

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 conceptual blueprint of a multi-tenant architecture. We conclude our paper with a discussion.

2. MULTI-TENANCY

Multi-tenancy is an organizational approach for SaaS applications. Although SaaS is primarily perceived as a busi-

ness model, its introduction has lead to numerous interesting problems and research in software engineering. Despite the growing body of research in this area, multi-tenancy is still relatively unexplored, despite the fact the concept of multi-tenancy first came to light around 2005[1].

While a number of definitions of a multi-tenant application exist [16, 17], they remain quite vague. 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. The ability of the application to share hardware resources.
2. The offering of a high degree of configurability of the software.
3. The architectural approach in which the tenants (or users) make use of a single application and database instance[2].

In the next two sections (2.1 and 2.2) we will demarcate multi-tenancy from two other organizational models, namely the multi-user and the multi-instance model. In Section 2.3 we will 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 heavily configure the application. 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 different for two tenants. An additional argument for the distinction is that the Service Level Agreement (SLA) of each tenant can differ [11], while this is usually not the case for users in a multi-user system.

2.2 Multi-Tenant versus Multi-Instance

Another contrasting approach is the multi-instance approach, in which each tenants gets his own instance of the application (and possibly also of the database). With the gaining in popularity of virtualization technology and cloud computing, the multi-instance approach is the "easier" way of creating multi-tenant like applications from a development perspective. Yet, the multi-instance approach is better suited if the number of tenants is likely to remain low [4], in part because the multi-instance model suffers from an increased maintenance cost. This increased maintenance cost can be attributed to the effort for deploying updates to numerous instances of the application.

[1]The Wikipedia entry was first created on November 14th, 2005; `http://en.wikipedia.org/wiki/Multitenancy`.

[2]Due to performance and/or legislative reasons, there might be more than one instance, but the number of instances should remain limited.

2.3 Key Characteristics of Multi-Tenancy

2.3.1 Hardware Resource Sharing

In traditional single-tenant software development, tenants usually have their own (virtual) server. This set-up is similar to the traditional Application Service Provider (ASP) model [12]. However, in the SME segment, server utilization in such a model is low. By placing several tenants on the same server, the server utilization can be improved [15, 16]. While this can also be achieved through virtualization, virtualization imposes a much lower limit on the number of tenants per server due to the high memory requirements for every virtual server [10]. 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, 9]:

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, 15]. These performance issues are caused by the fact that loading a database or table in memory, which is required for each tenant in the first and second variant, is an expensive operation. In the pure multi-tenancy variant, the shared table is loaded only once, resulting in a higher maximum number of tenants on the system.

2.3.2 High Degree of Configurability

In a single-tenant environment, every tenant has his own, (possibly) customized application instance. In contrast, in a multi-tenant setup, 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, just like in single-tenancy [12]. 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 [13], similar to software product line engineering [12].

Because of the high degree of configurability of multi-tenant software systems, it may be necessary to run multiple versions of an application (or parts of an application) next to each other. This situation might arise for reasons of backward compatibility or in situations where the legislation in a particular country changes. Because it is deemed undesirable to deploy different instances of a multi-tenant application, version support should be an integral part of a multi-tenant setup.

2.3.3 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 is runtime configurable.

This entails that in multi-tenancy 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, deployment is much easier and cheaper, particularly in the area of deploying the updates, as a the number of instances which are touched by the deployment action are clearly much lower.

In addition, new data aggregation opportunities are opened because all tenant data is in the same place. For example, user behaviour traces can be collected much easier, which can help to improve the user experience.

2.4 Benefits

From the previous paragraphs a number of reasons for companies to introduce multi-tenancy can be deducted:

1. Higher utilization of hardware resources. (§2.3.1)
2. Easier and cheaper application maintenance. (§2.3.3)
3. Lower overall costs, allowing to offer a service at a lower price than competitors. (§2.3.1, §2.3.3)
4. New data aggregation opportunities. (§2.3.3)

2.5 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, 4, 9] have described the possible variants of multi-tenancy, as we have described in Section 2.3.1. Wang et al. [15] 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 and the amount of data per tenant.

Kwok et al. [9] 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 [13] and Mietzner et al. [12] as well.

Guo et al. [4] 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 [5]. 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. [8] 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 [17].

3. CHALLENGES

Unfortunately, multi-tenancy also has its challenges and even though some of 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) [10]. 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 [4] very important.

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 [3].

3.5 Maintenance

In the typical evolutionary cycle of software, a challenge is formed by maintenance, e.g. adapting the software system to changing requirements and its subsequent deployment [6]. While it is clear that the multi-tenant paradigm can bring

[3] `http://ec.europa.eu/taxation_customs/taxation/vat/traders/invoicing_rules/article_1733_en.htm` (last visited on June 2nd, 2010)

serious benefits for deployment by minimizing the number of application and database instances that need to be updated, the situation for the actual maintenance is not so clear. In particular, introducing multi-tenancy into a software systems will add complexity, which will likely affect the maintenance process. Further research is needed to evaluate whether the hardware and deployment benefits outweigh the increased cost of maintenance.

4. MULTI-TENANT ARCHITECTURE CONCEPTUAL BLUEPRINT

When we started thinking how multi-tenancy affects an application, we came up with the architectural overview of Figure 1. Here we see that multi-tenancy affects almost all layers of a typical application, and as such, there is high potential for multi-tenancy to become a *cross-cutting concern*. To keep the impact on the code (complexity) low, the implementation of multi-tenant components should be separated from single-tenant logic as much as possible. If not, maintenance can become a nightmare because:

- Mixing multi-tenant with single-tenant code must be done in all application layers, which requires all developers to be reeducated about multi-tenancy.
- Mixing multi-tenant with single-tenant code leads to increased code complexity because it is more difficult to keep track of where multi-tenant code is introduced.

These two problems can be overcome by carefully integrating multi-tenancy in the architecture. In the remainder of this section, we describe the components of our architectural approach for implementing multi-tenancy as a cross-cutting concern.

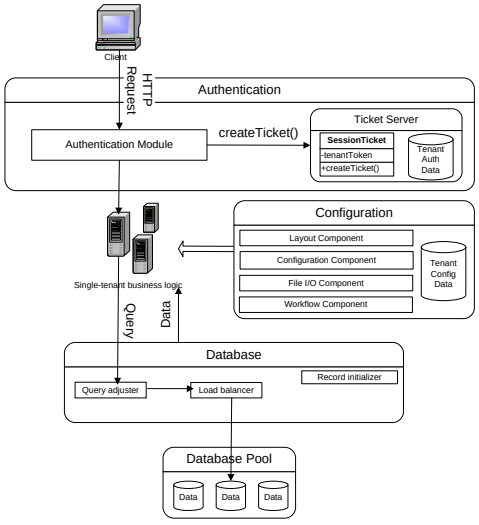

Figure 1: Architectural overview for multi-tenancy

4.1 Authentication

Because a multi-tenant application has only one application and database instance, all tenants use the same physical environment. In order to be able to offer customization of this environment and to make sure that tenants can only access their own data, tenants must be authenticated. While user authentication is possibly already present in the target application, a separate tenant-specific authentication mechanism might be required, for two reasons: (1) it is usually much easier to introduce an additional authentication mechanism, then to change the existing one, and (2) tenant authentication allows a single user to be part of more than one logical organization, which extends the idea of user authentication with "groups". A typical example of such a situation would be a bookkeeper, who works for multiple organizations.

4.2 Configuration

In a multi-tenant application customization must be made possible through configuration [13]. In order to enable multi-tenancy and let the user have a user-experience as if he were working in a dedicated environment, it is necessary to allow at least the following types of configuration:

Layout Style The layout style configuration component allows the use of tenant-specific themes and styles.

General configuration The general configuration component allows the specification of tenant-specific configuration, such as 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. For example, 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

In a multi-tenant application, there is a great requirement for data isolation. Because all tenants use the same database instance, it is necessary to make sure that they can only access their own data. Since current off-the-shelf DBMSs are not capable of dealing with multi-tenancy themselves [5], 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 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.

5. DISCUSSION

Because of the low number of instances, multi-tenancy sounds like a maintenance dream. Deployment of software updates becomes much easier and cheaper, due to the fact that a much smaller number of instances has to be updated. However, the complexity of the code does increase. In single-

tenant software, challenges like configuration and versioning are solved by creating a branch in the development tree and deploying a separate instance. In multi-tenant software, this is no longer acceptible, which means that features like these must be integrated in the application architecture, which inherently increases the code complexity and therefore makes maintenance more difficult.

We believe that multi-tenancy can be a maintenance dream, despite the increase in code complexity. However, the quality of the implementation is crucial. In a non-layered software architecture, the introduction of multi-tenancy can lead to a maintenance nightmare because of code scattering. On the other hand, in a layered architecture, it is possible to implement multi-tenancy as a relatively isolated cross-cutting concern with little effort, while keeping the application maintainable. This raises the expectation that, for maintenance in a multi-tenant application, refactoring a non-layered architecture into a layered one can be very beneficial.

Our architectural approach can guide the process of introducing multi-tenancy in an application. 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).

Future work includes evaluation of our architectural approach by enabling multi-tenancy in an existing industrial application [1].

Acknowledgements.

The authors would like to thank Exact for providing the funds and opportunity to perform this research. Further support came from the *NWO Jacquard ScaleItUp* project.

6. REFERENCES

[1] Cor-Paul Bezemer, Andy Zaidman, Bart Platzbeecker, Toine Hurkmans, and Aad 't Hart. Enabling multi-tenancy: An industrial experience report. In *26th IEEE Int. Conf. on Software Maintenance (ICSM)*. IEEE, 2010. To appear.

[2] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. Multi-tenant data architecture. `http://msdn.microsoft.com/en-us/library/aa479086.aspx`, June 2006.

[3] Alan G. Ganek and Thomas A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

[4] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao. A framework for native multi-tenancy application development and management. In *Proc. Int. Conf. on E-Commerce Technology (CEC) & Int. Conf. on Enterprise Computing, E-Commerce, and E-Services (EEE)*, pages 551–558. IEEE, 2007.

[5] Dean Jacobs and Stefan Aulbach. 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. März*, volume 103 of *LNI*, pages 514–521. GI, 2007.

[6] Slinger Jansen, Sjaak Brinkkemper, Gerco Ballintijn, and Arco van Nieuwland. Integrated development and maintenance of software products to support efficient updating of customer configurations: A case study in mass market ERP software. In *Proc. Int. Conf. Soft. Maintenance (ICSM)*, pages 253–262. IEEE, 2005.

[7] Jeffrey M. Kaplan. Saas: Friend or foe? In *Business Communications Review*, pages 48–53, June 2007. `http://www.webtorials.com/abstracts/BCR125.htm`.

[8] Thomas Kwok and Ajay Mohindra. Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications. In *Proc. Int. Conf. on Service-Oriented Computing (ICSOC)*, volume 5364 of *LNCS*, pages 633–648. Springer, 2008.

[9] Thomas Kwok, Thao Nguyen, and Linh Lam. A software as a service with multi-tenancy support for an electronic contract management application. In *Proc. Int. Conf. on Services Computing (SCC)*, pages 179–186. IEEE, 2008.

[10] Xin Hui Li, Tiancheng Liu, Ying Li, and Ying Chen. SPIN: Service performance isolation infrastructure in multi-tenancy environment. In *Proc. Int. Conf. on Service-Oriented Computing (ICSOC)*, volume 5364 of *LNCS*, pages 649–663, 2008.

[11] Hailue Lin, Kai Sun, Shuan Zhao, and Yanbo Han. Feedback-control-based performance regulation for multi-tenant applications. In *Proc. Int. Conf. on Parallel and Distributed Systems (ICPADS)*, pages 134–141. IEEE, 2009.

[12] Ralph Mietzner, Andreas Metzger, Frank Leymann, and Klaus Pohl. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *Proc. of the ICSE Workshop on Principles of Eng. Service Oriented Systems (PESOS)*, pages 18–25. IEEE, 2009.

[13] Nitu. Configurability in SaaS (software as a service) applications. In *Proc. of the 2nd annual India softw. eng. conf. (ISEC)*, pages 19–26. ACM, 2009.

[14] Chang-Hao Tsai, Yaoping Ruan, Sambit Sahu, Anees Shaikh, and Kang G. Shin. Virtualization-based techniques for enabling multi-tenant management tools. In *18th IFIP/IEEE Int. Workshop on Distr. Systems: Operations and Management (DSOM)*, volume 4785 of *LNCS*, pages 171–182. Springer, 2007.

[15] Zhi Hu Wang, Chang Jie Guo, Bo Gao, Wei Sun, Zhen Zhang, and Wen Hao An. 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)*, pages 94–101. IEEE, 2008.

[16] Bob Warfield. 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 June 2nd, 2010), October 2007.

[17] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *Proc. of the 35th SIGMOD int. conf. on Management of data (SIGMOD)*, pages 889–896. ACM, 2009.

TUD-SERG-2010-031
ISSN 1872-5392