Multi-Tenant Architecture Comparison

Jaap Kabbedijk, Michiel Pors, Slinger Jansen, and Sjaak Brinkkemper

Department of Information and Computing Sciences
Utrecht University, Netherlands
{J.Kabbedijk, M.Pors, Slinger.Jansen, S.Brinkkemper}@uu.nl

Abstract. Software architects struggle to choose an adequate architectural style for multi-tenant software systems. Bad choices result in poor performance, low scalability, limited flexibility, and obstruct software evolution. We present a comparison of 12 Multi-Tenant Architecture (MTA) patterns that supports architects in choosing the most suitable architectural pattern, using 17 assessment criteria. Both patterns and criteria were evaluated by domain experts. Five architecture assessment rules of thumb are presented in the paper, aimed at making fast and efficient design decisions. The comparison provides architects with an effective method for selecting the applicable multi-tenant architecture pattern, saving them effort, time, and mitigating the effects of making wrong decisions.

Keywords: multi-tenancy; architecture patterns; quality attributes;

1 Introduction

As a consequence of the current shift of on-premises software to the cloud [4], software architects find themselves facing numerous new challenges related to the adequacy of architectures for cloud software. A commonly used technique in architecting for Software-as-a-Service (SaaS) is the use of the concept of multi-tenancy, which is defined for this research as *"a property of a system where multiple varying customers and their end-users share the system's services, applications, databases, or hardware resources, with the aim of lowering costs"* [11].

Multi-tenancy can bring about many benefits. By serving the software service from a centrally hosted location, clients are relieved from the responsibility of purchasing and maintaining expensive in-house servers. The total cost of ownership decreases, giving the SaaS provider access to new potential customers that previously could not afford the expenses [2]. In addition, the utilization rate of hardware in a multi-tenant environment is higher than in a single-tenant environment [12]. Furthermore, when multiple customers share application and data instances, the total number of running instances will be lower than in a single-tenant environment, catering the same number of customers. A low number of instances is beneficial for maintenance [9] and is beneficial for application development [1].

However, multiple barriers withhold service providers from massively switching to multi-tenant environments. The challenges for multi-tenancy adoption include performance [10], scalability, security [7], and the re-engineering of current software applications [13]. Selecting the appropriate multi-tenant architecture is a complex problem due to the existence of numerous alternative architectural patterns. Benefits and barriers of multi-tenancy are identified and described in literature, but the aspect of choosing an appropriate multi-tenant architecture based on software vendors' preferences has received little attention in literature. Finding the most suitable multi-tenant architecture is crucial; it expresses a fundamental structural organization schema for a provider's software system. However, choosing the appropriate architecture is a wicked problem [5]. Accounting for all the challenges and benefits complicates the decision process considerably [8].

This paper presents a comparison of different Multi-Tenant Architecture (MTA) patterns, based on the the mixed method research approach used within this study (Section 2). The twelve different MTA patterns are shown in section 3, together the MTA comparison matrix in section 4. We conclude with a discussion on the comparison, together with threats to validity present and future work in section 5, focussing on the importance of evaluating more effective methods in architectural decision making.

2 Research Approach

The main research question of this research is formulated as follows:

RQ. *How can a SaaS provider be supported in the decision process of choosing an applicable multi-tenant architecture?*

Three sub questions are answered in order to develop a decision model that solves the main research question. The decision model consists of three fundamental elements, which need to be identified. The first element is a set of multi-tenant architectures to choose from. Hence, the first sub question is defined as follows:

SQ1. *What distinctive layers in multi-tenant architectures can be defined? —* Using a Structured Literature Research (SLR), the distinctive layers in multi-tenant architectures are identified in SQ1. For more details on the search query, criteria, strategy and construction of trail searches, please see [11]. Instead of searching directly for multi-tenant architectures and documenting them, a different approach is taken. First, different layers on which multi-tenancy can be applied are identified. Then, generic multi-tenant architectures are identified, based on these layers. The list of identified architectures is evaluated by domain experts to ensure the list is complete and concise. The expert evaluation is not only essential for checking the correctness of the list, but also to make sure the identified architectures reflect *relevant* and *implementable* architectures.

SQ2. *What are the relevant decision criteria for choosing an appropriate multi-tenant architecture? —* SQ2 aims at identifying the different decision criteria, or architecturally significant requirements, related to multi-tenant architectures.

The decision criteria are quantifiable attributes distinguishing between the different multi-tenant architectures. Similar to the identification of the MTAs, a structured literature research is carried out to identify the list of criteria. The identification process results in a large set of criteria, which is analyzed in order to merge similar and delete unimportant attributes. Consequently, the completeness and conciseness of the list is evaluated in an expert evaluation. Finally, the multi-tenant architectures must be evaluated using the decision criteria, resulting in performance scores. The final sub question is stated as:

SQ3. *How do the different multi-tenant architectures perform on the decision criteria?* — In SQ3, an evaluation is performed in which all MTAs are assessed by domain experts on the identified decision characteristics. The evaluation serves as a basis for MTA decision making.

3 Multi-tenant Architectures

The levels at which multi-tenancy can be applied, resulting from the literature study, are shown in Table 1. All levels are listed together with the frequency of occurrence (N) in literature. The different levels are depicted as layers in a stack with decreasing granularity from top to bottom in Figure 1. The granularity aspect translates to a sharing versus isolation continuum, where the lowest layer has the lowest level of sharing with the highest level of isolation. For the highest layer it is vice versa. When multi-tenancy is applied at a specific level, the levels below that level are shared among tenants as well, but isolation occurs at the levels above, i.e. for each tenant a dedicated instance is running. This applies to the application and data layer independently.

Multi-tenancy level	N
Application Instance	16
Database Server	16
Database	15
Operating System	15
Hardware	14
Schema	14
Middleware	12
Virtual Machine	9
Application Server	4

Table 1. Multi-Tenancy levels identified in literature

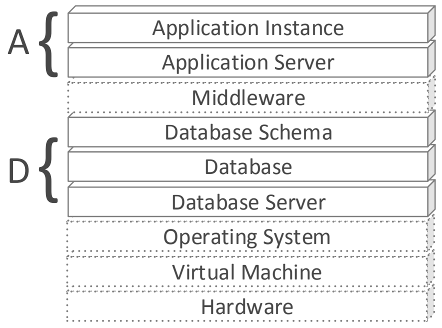

Fig. 1. Multi-tenancy computing stack. *'A' and 'D' relatively indicate the Application and Data related layer sets*

The final two levels of the stack in the data layer are the *database* and *schema level*. These two levels were first described by Chong et al. [3]. When tenants are consolidated in a single database, each tenant operates its own set of tables. In schema-level multi-tenancy, isolation occurs at table row level.

The *application* related layer set (A) and the *data* related layer set (D) are stacks commonly used in enterprise architecture in order to separate concerns [6]. Within this research the application layers and data layers are identified as separate *layer sets*, each containing different sub layers, as can be seen in Figure 1.

Consequently, three tenancy levels, indicated by a two letter abbreviation, are identified in the *Application* related layer set (**A**). The different levels result from identifying ascending levels of sharing among all layers on the set:

1. **AD** - A Dedicated Application server is running for each tenant, and therefore each tenant receives a dedicated application instance.
2. **AS** - A single Application Server is running for multiple tenants and each tenant receives a dedicated application instance.
3. **AI** - A single application server is running for multiple tenants and a single Application Instance is running for multiple tenants.

The first level corresponds to multi-tenancy enabled at the hardware or virtual machine level. The second level is equal to application server multi-tenancy. The third level is the same as multi-tenancy enabled at the application instance level. In the *Data* related layer set (**D**) a service provider can select one the following four tenancy levels:

1. **DD** - A Dedicated Database server is running for each tenant, and therefore each tenant receives a dedicated database.
2. **DS** - A single Database Server is running for multiple tenants and each tenant receives a dedicated database.
3. **DB** - A single DataBase server is running for multiple tenants, data from multiple tenants is stored in a single database, but each tenant receives a dedicated set of tables.
4. **DC** - A single database server is running for multiple tenants, data from multiple tenants is stored in a single database and a single set of tables, sharing the same Database sChema.

The first level is equal to multi-tenancy applied at the hardware or virtual machine level. The second one corresponds to database server multi-tenancy. The third alternative is the same as multi-tenancy applied to the database and the final one is equal to database schema multi-tenancy.

From these options in both the application and data layer, the set of multi-tenant architectures (MTAs) are constructed. Based on the tenancy levels within the layers, the number of possible architectures is twelve. Because all MTAs prescribe a specific tenancy level in set A and D, each architecture is defined as a tuple:

$$MTA = \langle \{AD, AS, AI\}, \{DD, DS, DB, DC\} \rangle \tag{1}$$

Each of the twelve MTAs can be seen as an architectural pattern in which tenants (Tenant A, B and C in the example MTAs) communicate with a software application consisting of an application layer and a data layer as shown in Figure 2. For a complete overview of all MTAs please see [11].

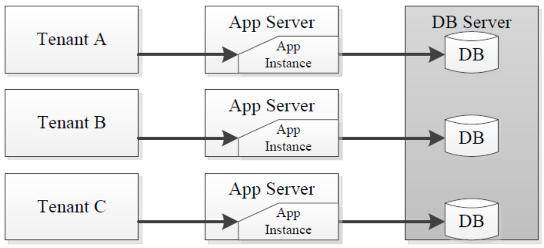

Fig. 2. MTA$\langle AD, DS\rangle$ - Dedicated Application Server & Shared Database Server

In Figure 2 the application layer is represented as a set of application servers running one or multiple application instances. The data layer is displayed as a set of database servers, running one or more databases, in which one or multiple database schema's exist. If one of these entities is shared among the tenants, its color is gray. If its dedicated to only one tenant, its colored white. For the sake of simplicity only three tenants are displayed in the architectures. A service provider can offer his software application to more than three tenants, the patterns merely presents possible arrangements of shared resources.

4 MTA Comparison Matrix

The MTA pattern comparison offers decision makers a method to make an informed and balanced decision on the MTAs to consider implementing for their software product. The MTA Comparison Matrix in Table 2 offers a high level of detail, while also giving a quick overview of the strengths and weaknesses of all patterns. Using the matrix, architects can get an overview of the consequences of all different MTA patterns and assess the weight of the consequences for their specific situation. Based on the consequences and the weights, architects can select a subset of patterns to evaluate more thoroughly. To help in selecting a subset for future analysis, this section presents some Rules of Thumb (RT) derived from the comparison matrix and are helpful in giving decision makers a quick overview of the most important consequences of an MTA assessment.

RT1. Focus on the database dimension — The effect of different MTAs on decision criteria is largest on the database dimension. The MTA Comparison Matrix shows the effect of database related decisions is higher than application related decisions. Choosing between a set of MTAs, focus on database related decisions first, and application related decisions after.

RT2. Sharing database tables enables serving of many tenants but harms robustness — Selecting an MTA in which the database schema is

Decision Criterion	$\langle AD, DD\rangle$	$\langle AS, DD\rangle$	$\langle AI, DD\rangle$	$\langle AD, DS\rangle$	$\langle AS, DS\rangle$	$\langle AI, DS\rangle$	$\langle AD, DB\rangle$	$\langle AS, DB\rangle$	$\langle AI, DB\rangle$	$\langle AD, DC\rangle$	$\langle AS, DC\rangle$	$\langle AI, DC\rangle$	Dist. Factor (σ^2)
Time Behavior	5.0	4.0	4.0	4.0	4.0	3.0	4.0	3.5	3.0	3.5	3.0	2.0	0.6
Resource Utilization	2.5	2.5	3.0	2.5	3.0	3.0	3.0	3.0	4.0	3.0	3.0	4.5	0.4
Throughput	4.5	3.0	3.0	4.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	0.2
Number of Tenants	1.0	3.0	3.0	3.0	3.5	4.0	3.0	4.0	4.0	3.0	4.0	5.0	1.0
Number of End-Users	2.5	3.5	3.0	3.0	3.5	3.5	3.0	3.5	4.0	3.5	4.0	4.0	0.2
Availability	4.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	0.1
Recoverability	5.0	4.5	4.5	4.0	4.0	4.0	3.0	3.0	3.0	2.0	2.0	2.0	1.1
Confidentiality	5.0	4.5	4.0	4.0	4.0	4.0	3.5	3.0	3.0	2.0	2.0	2.0	1.0
Integrity	4.5	4.0	3.0	4.0	3.5	3.0	3.5	3.0	3.0	3.0	2.5	2.5	0.4
Authenticity	4.5	3.5	3.0	3.5	3.0	3.0	4.0	3.0	3.0	3.0	3.0	3.0	0.2
Maintainability	1.5	2.5	3.0	2.0	3.0	3.5	2.5	4.0	4.0	3.0	4.0	5.0	1.0
Portability	5.0	5.0	5.0	4.5	4.5	4.5	4.0	4.0	4.0	3.0	3.0	3.0	0.6
Deployment Time	1.5	3.0	3.0	2.5	3.5	4.0	3.0	4.0	4.0	3.0	4.0	5.0	0.8
Variability	5.0	4.0	2.0	5.0	4.0	2.0	4.5	3.5	2.0	2.5	2.0	1.0	1.9
Diverse SLA	5.0	4.0	3.0	4.0	3.5	2.5	4.0	3.0	3.0	3.0	2.5	2.0	0.7
Software Complexity	5.0	4.5	4.0	4.5	4.5	3.5	4.0	4.0	3.0	2.5	2.5	2.0	0.9
Monitoring	1.0	2.5	3.0	2.5	3.0	3.0	3.0	4.0	4.0	3.0	4.0	5.0	1.0

Table 2. Multi-Tenant Architecture Comparison Matrix (In color) - **1.0** indicates a highly negative effect, by applying the pattern, on the decision criterion. **5.0** indicates a highly positive effect.

shared (i.e. $\langle A?, DC\rangle^1$) is beneficial if the software product serves many tenants and end-users. The product is easy to maintain and monitor, and deployment time is minimal. The recoverability of the system, on the other hand, is greatly compromised. It is difficult to implement variability and tenant data may be at risk of unintentional sharing. Based on this trade-off, SaaS providers should select $\langle A?, DC\rangle$ when designing a large scale software product with limited variability requirements.

RT3. Sharing application instances helps maintainability and performance but harms variability — Choosing an MTA, decision makers can decide to share the application instance among tenants (i.e. $\langle AI, D?\rangle$). Doing so causes the maintainability and ease of monitoring to increase. Also the resource utilization is better and the deployment time low. The variability of the software product, however, is lower and more difficult to implement. Because of this, SaaS should choose $\langle AI, D?\rangle$ when maintainability and performance efficiency are important.

^{1} '?' is used as a single character wild card.

RT4. Ease of implementing variability differs greatly per MTA — Out of all decision criteria, variability has the highest distinction factor. This means the variability of a software product is for a significant part determined by the implemented MTA. Choosing an MTA with a low tenancy level (i.e. $\langle AD, DD\rangle$, variability is relatively easy to achieve. Selecting an MTA with a high tenancy level however (i.e. $\langle AI, DC\rangle$), causes large problems implementing variability over all tenant instances.

RT5. Dedicated servers improve performance and variability, but hamper scalability — When choosing an MTA with dedicated servers (i.e. $\langle AD, DD\rangle$) the time behavior, recoverability, variability and confidentiality are expected to be good, and software complexity low. The downside to this approach is the low scalability of the system; when the number of tenants increases, dedicated servers become hard to maintain and hardware costs will rise. Choose $\langle AD, DD\rangle$ for software products with a small user base that need to have high performance and a high level of flexibility. Typically large enterprise applications fall in this category.

The rules of thumb listed in this section do not aim for completeness, but rather give software architects and decision makers a collection of rules to guide their architecture selection.

5 Discussion and Conclusion

The identification of the 12 different multi-tenant architecture patterns and the comparison of the patterns, along with a list of assessment criteria and rules of thumb, support SaaS providers in providing a concise and versatile method for multi-tenant architecture assessment. In case specific assessment criteria or MTAs are irrelevant to a decision maker for some reason, those elements can be easily removed from the analysis, simplifying the selection of a suitable architecture. If a SaaS provider feels important decision criteria are missing from the assessment model, extra decision criteria can be added in the analysis. However, performance values of the MTAs on these criteria can not be provided in this research.

We identify the following threats to validity to this study: 1. The small sample of 10 domain experts used may lead to biased results. A larger set would potentially increase the generalizability of the results. 2. All experts are from the same company. This threat is mitigated by the fact they are all employed at different independent projects. 3. The comparison matrix is not evaluated in practice in an extensive case study to test the applicability. By performing a case study, the appropriateness of the matrix can be validated.

All are threats to *external* validity, as defined by Yin [14]. We suggest further research to focus on demonstrating the analytic hierarchy process in conjunction with the comparison matrix at several companies. Then, the ratings can be evaluated more thoroughly resulting in possible adjustments for these performance values. Furthermore, the ratings provided in this research are based on subjective judgements of ten experts. The accuracy of the ratings can be increased by surveying a larger number of experts, causing a decrease of the standard deviation.

6 Acknowledgments

This research is funded by the NWO/ICT-Regie 'Product as a Service' grant. Special thanks to Leen Blom and the experts willing to cooperate in our research. We would also like to thank Hans van Vliet for providing valuable feedback.

References

1. Bezemer, C.P., Zaidman, A., Platzbeecker, B., Hurkmans, T., t Hart, A.: Enabling multi-tenancy: An industrial experience report. In: Proc. of the Int. Conference on Software Maintenance (ICSM). pp. 1–8. IEEE (2010)
2. Chong, F., Carraro, G.: Architecture strategies for catching the long tail. Tech. rep., MSDN Library, Microsoft Corporation (2006)
3. Chong, F., Carraro, G., Wolter, R.: Multi-tenant data architecture. Tech. rep., MSDN Library, Microsoft Corporation (2006)
4. D'souza, A., Kabbedijk, J., Seo, D., Jansen, S., S., B.: Software-as-a-service: Implications for business and technology in product software companies. In: Proceedings of the Pacific Asia Conference on Information Systems (PACIS). pp. 140–146 (2012)
5. Esfahani, N., Razavi, K., Malek, S.: Dealing with uncertainty in early software architecture. In: Proc. of the Int. Symposium on the Foundations of Software Engineering. p. 21. ACM (2012)
6. Fowler, M.: Patterns of enterprise application architecture. Addison-Wesley Professional (2003)
7. 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 Int. Conference on E-Commerce Technology (CEC). pp. 551–558. IEEE (2007)
8. Kazman, R., Asundi, J., Klein, M.: Quantifying the costs and benefits of architectural decisions. In: Proc. of the Int. Conference on Software Engineering (ICSE). pp. 297–306. IEEE Computer Society (2001)
9. 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. Conference on Services Computing (SCC). pp. 179–186 (2008)
10. Lin, H., Sun, K., Zhao, S., Han, Y.: Feedback-control-based performance regulation for multi-tenant applications. In: Proc. of the Int. Conference on Parallel and Distributed Systems (ICPADS). pp. 134–141. IEEE (2009)
11. Pors, M., Blom, L., Kabbedijk, J., Jansen, S.: Sharing is caring - a decision support model for multi-tenant architectures. Tech. Rep. UU-CS-2013-015, Department of Information and Computing Sciences, Utrecht University (2013)
12. Sääksjärvi, M., Lassila, A., Nordström, H.: Evaluating the software as a service business model: From cpu time-sharing to online innovation sharing. In: Proce. of the Int. Conference e-Society. pp. 27–30. Qawra, Malta (2005)
13. Tsai, C.H., Ruan, Y., Sahu, S., Shaikh, A., Shin, K.G.: Virtualization-based techniques for enabling multi-tenant management tools. In: Managing Virtualization of Networks and Services, pp. 171–182. Springer (2007)
14. Yin, R.K.: Case study research: Design and methods, vol. 5. Sage (2009)