Multitenancy in SaaS: A comprehensive Survey

Pallavi G B, Dr. P Jayarekha

Abstract— Cloud computing is a technology which provides enterprise-grade computing resources as services to customers through internet. One of the popularly available services is software viz., software as a service (SaaS). Many of the SaaS providers make use of a multitenant model to host their applications. Multitenancy is an architectural approach in which a single instance of a software application serves multiple customers referred as tenants, where one application is run on a single database instance for multiple organizations. However, since sharing of a single instance of software leads to distribution of underlying hardware resources among tenants, a multitenant platform poses challenges with respect to architectural, implementation and security issues. In this paper, we have made an extensive survey to understand the various aspects of these issues. The study results can be used not only to identify advantages and disadvantages of these aspects, but also to identify areas requiring future research.

Index Terms— Architecture, Implementation, Multitenancy, SaaS, Security, Metadata, Survey

1 INTRODUCTION

Cloud Computing has become an important computing technology. Bhaskar Prasad et al., [1] in their survey states that cloud computing has emerged as a popular computing model to support large volumetric data using clusters of commodity computers.

According to Peter Mell et al., NIST [16], cloud computing is a model for enabling convenient, on-demand access through internet to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. The users can access these services available, without having any previous know-how on managing the resources involved.

One of the important services available today on cloud computing paradigm is software as a service. SaaS is a novel delivery model of software that provides an application for multiple users via internet as a form of 'on demand service'[2].This is an attractive offer for software companies as they can use various IT services without the need to purchase and maintain their own IT infrastructure[15]. Also, service provider achieves full economy of scale by hosting such SaaS application using a multitenant model.

The authors Corl-Paul Bezemer and Andy Zaidman in [15] positions multitenancy as one of the key concerns in SaaS. They describe multitenancy as an architectural principle that enables SaaS application to serve multiple client organizations (tenants) using a single service instance.

- *Pallavi G B, Assistant Professor, Dept of CSE,BMSCE is currently pursuing Ph.D. in the field of Multitenancy in SaaS in VTU,India. E-mail: pallavi.cse@bmsce.ac.in*
- *Dr. P Jayarekha, Associate Professor,Dept of ISE,BMSCE holds Ph.D. degree in computer science. She has published more than 15 research papers in referred International Journals and also few in national and international conferences. Research Scholars are working on various fields like cloud computing, WSN and related areas under her guidance.E-mail: jayarekha.ise@bmsce.ac.in*

Multitenancy invariably occurs at the database layer of the SaaS application[5] i.e., customer share the same hardware resources by using a single shared application and database instance, while at the same time tenants are isolated from one another as if they were running on physically segregated resources

One of the main advantages of a multi tenant application is the operational benefits [22]. Since all the application code is in one place, it is much easier to maintain, update and backup the application and its data. Another advantage of multi tenancy is the lower system requirements. Because an application and database are shared by multiple clients, it is not necessary to have a dedicated server for every client. Resource utilization is near optimal and customers can avoid under/over provisioning of resources.

However, multitenant application poses several challenges and difficulties as well. Some of these challenge stated by authors of [15] are as follows:

Performance: Equal amount of resources assigned to each tenant may lead to very inefficient utilization of resources and is therefore undesirable in a multitenant system.

Scalability: Because all the tenants share the same application and data store, scalability has become one of the major issues. Addition to that, tenants from various geographical locations may use an application, which can have an impact on scalability requirement.

Security: Security is one of the key challenges of multitenancy, as a security breech can result in exposure to data of other tenants. So data protection is an important issue in multitenancy.

In order to meet such challenges multitenant architectural, implementation and security related research are being carried out.

This paper describes a comparative study of these different aspects of multitenancy. The paper is organised as follows: Section 2 introduces the architectural issues of multitetancy, Section 3 briefs about various implementation details of multitenancy and Section 4 defines security concerns associated with multitenancy, finally Section 5 concludes the paper.

2 ARCHITECTURAL ISSUES OF MULTITENANCY

As mentioned, multitenancy is an architectural approach that benefits both the multitenant application provider and users. Various architectural approaches have been devised as a result of research on this principle.

2.1 Architectural overview for multitenancy

An architectural overview for multitenancy has been presented by the authors of [15] as shown in Fig 1. They argue that multitenancy could become a cross cutting concern in any SaaS application as it affects almost all layers of a typical application.

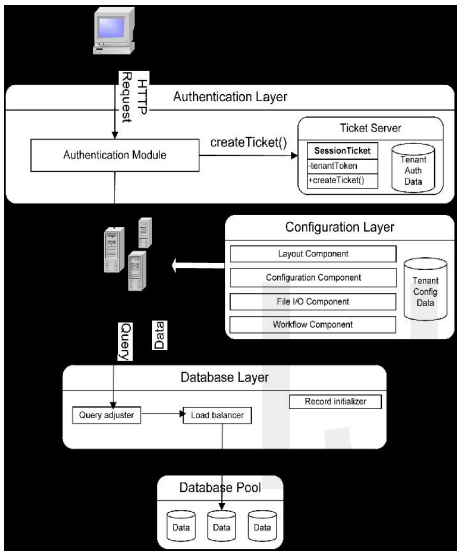


Fig1. Architecture overview for multitenancy [15]

In Fig 1, a tenant –specific authentication mechanism is shown in order to authenticate tenants to access only their own data. Further configuration layer facilitates tenants to make their own application specific customization like changing layout style, creating workflow etc so that user gets a feel as if he were working in a dedicated environment. The database layer is of at most importance in any multitenant environment as all the tenants make use of the same database. Hence data isolation among tenants is a major requirement. But because most of the current off-the –shelf DBMSs are not capable of dealing with multitenancy a layer between business logic and applications database pool could handle tasks like creation of new tenants in the database, query adoption and load balancing.

2.2 Types of Architecture

As per Frederick Chong et a1, [8] there are mainly three kinds of architecture of multitenancy that are applicable as for as database sharing among tenants are concerned.

1. Shared application, separate database: Each tenant obtains a private database instance while sharing a single machine.

Fig 2 [8]

2. Shared application, shared database, separate table: Each tenant obtains a private set of tables while sharing a single database instance

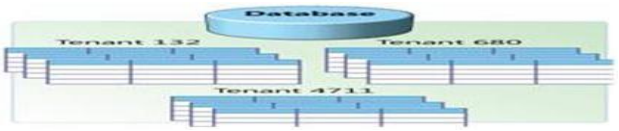
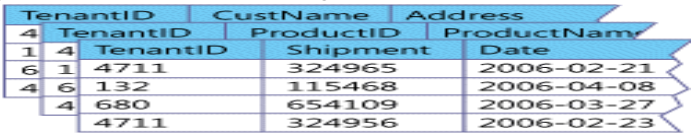


Fig 3 [8]

3. Shared application, shared table: Each tenant shares a single database instance and single set of tables.



TenantID	CustName	Address	
4	TenantID	ProductID	ProductNam
1	4	TenantID	Shipment	Date
6	1	4711	324965	2006-02-21
4	6	132	115468	2006-04-08
	4	680	654109	2006-03-27
		4711	324956	2006-02-23

Fig 4 [8]

2.3 Metadata-driven architecture of multitenancy

Craig D Weismann et al., in [3] mentions that a multitenant application can fulfil the requirement of multiple customers by using the hardware resources and staff needed to manage just a single software instance [Fig 5].

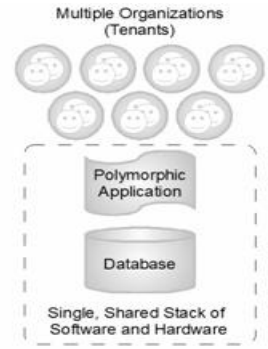


Fig 5 A multitenant application shares a single stack of resources to satisfy the needs of multiple organization [3]

However the authors in [3] describe how Force. Com, one of the leading titans of SaaS, has adopted metadata-driven architecture to achieve multitenancy. As per the authors, in Force.com, everything exposed to developers and application users is internally represented as metadata. When developers write custom application, defines custom table, write some procedural code, Force.com does not create an actual table in a database or compile any code. Instead, Force.com simply stores metadata that the platform's engine can use to generate the "virtual" application components at runtime. When some-

one wants to modify or customize something about the application, all that's required is a simple non-blocking update to the corresponding metadata .

The authors also stress upon the need of a multitenant application that allow tenants to

- To create custom extensions
- To keep data secured in a shared environment
- To customize the application interface and business logic without affecting functionality and availability
- To upgrade applications code base without breaking tenant specific customization.

In order to attain these characteristics the authors suggests that multitenant application must be dynamic or polymorphic which is achieved by adopting a runtime engine that generates application component from metadata-data about application itself. Hence this kind of architecture is termed as metadata driven architecture [Fig 6].

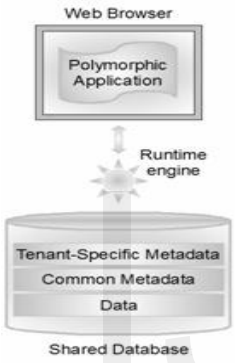


Fig 6 Metadata driven architecture [3]

Sungjoo Kang et al., in [4] have adopted this approach while designing architecture for a multitenant SaaS application platform as shown in Fig 7. This is a three tier architecture which provides an execution environment on which business SaaS applications can operate on.

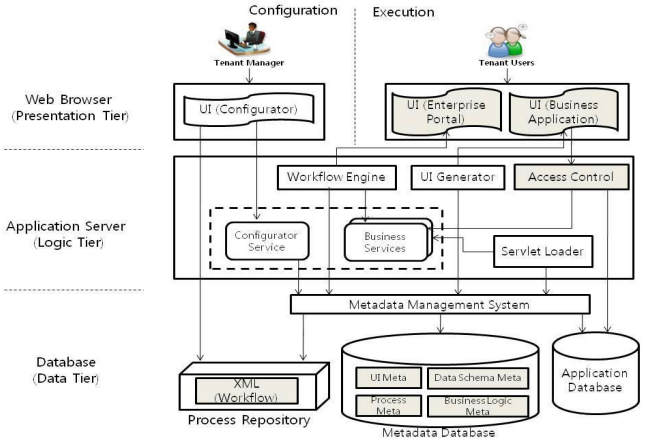


Fig 7 Conceptual architecture for SaaS platform [4]

As per the study of this paper:

- The platform is composed of several key components like Configurator, Runtime Engine and Metadata Management System. The configured aspects of SaaS application by the tenant manager are stored as metadata in metadata database. Codebase developed by application developers is stored in application database. Runtime engine generates tenant specific application using codebase and metadata.
- Metadata management system provides two key features for supporting multitenancy.
  - The first one is an access control for supporting multiple tenants which is accomplished by using metadata API's for Logical Tier. When the web browser sends request to the application server, business logic access to database with metadata API. Then, Metadata management system converts the request to the optimized query to retrieve tenant specific UI pages and data.
  - The other feature is providing extended fields for data object. As each tenant would require additional data to be stored in the database, they might have to add extra fields along with the fields provided in the database. This is achieved by adding the metadata information about the additional fields into the metadata database of custom fields which will be used to extract the specific extended fields of specific tenants at runtime.

Steve Bobrowski in [12] stipulates such meta-data driven multitenant database approaches as organic Multitenancy. As per the author, Organic Multitenancy can be described as an appcentric resource sharing model – in which multitenancy is woven into the fabric of the cloud from ground up. The author also gives example of Database.com, an underlying organic multitenant database that powers Force.com and which makes use of a runtime engine that generates all application components from metadata.

3 IMPLEMENTATION OF MULTITENANCY

Dean Jacobs and Stefan Aulbach describes and compares the three approaches of implementing multi-tenant databases : shared machine, shared process, and shared table[7] which corresponds to the three architectures described by Frederick Chong et a1 in [8]. Within the scope of the experiments they conducted, they list their observations about each approach as follows –

1. Shared Machine

- It does not require modifications of implementation of database
- It does not reduce customer isolation
- Memory pooling does not take place.
- Each database requires its own connection pool on each application server and sockets are not shared among customers.

2. Shared Process
- Pooling of memeory and scalability is achieved in a better way.
- Customers can share memory and connection pools.
- However, scalability can be further improved by adopting the third appraoch that is shared table approach.

3. Shared Table
- Its best at pooling resources
- Administrative operations can be executed in bulk.
- Connection pools can be shared by customers.
- Migration is hard because the files on disk have intermingled data from multiple customers.
- Performance is an issue since this intermingling may spread out across many pages.
- Security will be a major concern as connection pools and data are shared.

In order for the tenants to store data in a single set of tables, several storage models are prescribed so that all tenants data is stored in one set of tables and when a particular tenanat reqests for a data, only his requested data is provided and all the other tenant's data is hidden .

3.1 Storage Models

In [5], Stefan Aulbach and team discussed about various storage models used to implement multitenant databases of shared table. The different models termed as schema mapping techniques and their working as per the authors are:

- Basic layout
- Private table layout
- Extension table layout
- Generic Structures
  - Universal table layout
  - Pivot table layout
  - Chunk table layout

I. The basic technique for implementing multi-tenancy is to add a tenant ID column to each table and share tables among tenants.This technique is used by simple SaaS services like Email etc. However, this approach is a good way to consolidate various tenants data to be stored on the same database. It's suitable for tenants who do not need extensions on the data fields they operate on as this approach does not provide extensibility.

Basic layout Table 1

Account		
TenId	AcctId	Name
22	1	Arya
22	2	Girish
37	1	Bipin
45	1	Bhaskar

II. The next approach is called private table layout which can be adopted which supports extensibility. In this approach, each tenant will be given their own private table.The metadata related to these private tables are managed by database, there is no overhead of maintaining the metadata in the private tables of tenants. However, as separate tables are maintained on a per tenant basis, this approach provides only moderate consolidation of these tables.

Private Table layout

Table2a

Account22			
Aid	Name	Hospital	Beds
1	Arya	Apollo	120
2	Girish	St.John	1255

Table 2b

Account35	
Aid	Name
1	Bipin

Table 2c

Account 42		
Aid	Name	Dealers
1	Bhaskar	70

Hence, a better approach is to combine the above two mentioned layouts to obtain better consolidation and extensibility. Viz ., all the tenant's common data is stored in a single base table while their corresponding extension fields are stored and maintained in separate extension fields in extension tables as shown in Table 3a,3b and 3c. But, because multiple tenants may use the same extension tables, the extension tables and the base tables should be given a tenant ID column .Also inorder to reconstruct a logical source table corresponding to a particular tenant a row column needs to be added against which join conditions will be applied. This in turn adds the overhead of adding metadata in the data table itself as shown

Extension table layout
Table 3a

Account$_{Ext}$			
TenId	Row	AcctId	Name
22	0	1	Arya
22	1	2	Girish
37	0	1	Bipin
45	0	1	Bhaskar

Table 3b

Health care $_{Account}$			
Tenant	Row	Hospital	Beds
22	0	Apollo	120
22	1	St.John	1255

Table 3c

Automotive$_{Account}$		
Tenant	Row	Dealers
45	0	70

Apart from these approaches one of the methods widely adopted to implement multitenant data base is to use generic-structures.These generic structures do not place a limitation on consolidation or exensibility.The logical tables could be mapped to fixed generic structures. The various kinds of generic structures are:

→Universal Table Layout: A Universal Table is a generic structure with a Tenant column, a Table column and a large number of generic data columns. The data type of these generic columns will be VARCHAR into which the other datatypes can be converted. Here the n-th column of each logical source table for each tenant is mapped into the n-th data column of the Universal table so that different tenant can extend the same table in different way.

The advantage of this layout is that there is no need to reconstruct the logical source table as all the data are stored in the same physical universal table.Hence data needs to be selected from one singletable.

The disadvantage is that the rows needs to be very wide as it has to accommodate all possible data fields of all tenants and comprises of more null value for the obvious reason that all tenants need not use all data fields of the table.

Universal table layout Table 4

Universal							
Tenant	Table	Col1	Col2	Col3	Col4	Col5	Col6
22	0	1	Arya	Apollo	120	,---	,---
22	1	2	Girish	St.John	1255	,---	,---
37	0	1	Bipin	,---	,---	,---	,---
45	0	1	Bhaskar	70	,---	,---	,---

→Pivot Table Layout: In this type each field type(int, str etc) of the logical table is stored in a separate pivot table.Hence the more the number of data types of tenants the more the pivot tables. Each field of each row in a logical source table is given its own row in the pivot table. Hence each pivot table will have Tenant, Table and Row columns as shown in the Table 5a, 5b and 5c.The Col column in these tables specifies which source field a row represents and a single data bearing coulmn for the value of that field.

The advantage of this generic approach is that the number of null values of data fields are minimal unlike universal table. Type safety is also achieved since generic data types like varchar is avoided.T

The disadvantage is that more number of joins has to be executed to reconstuct the logical table as a result of a query execution. Also the actual physical table contains most of the meta data stored in it used for reconstruction of thelogical tables.

Pivot table layout

Table 5a

Row 0 →

Account17			
Aid	Name	Hospital	Beds
22	Arya	Apollo	120
22	Girish	St.John	1255

Table 5b

Pivotint				
Tenant	Table	Col	Row	Int
22	0	0	0	1
22	0	3	0	120
22	0	0	1	2
22	0	3	1	1255
37	1	0	0	1
45	2	0	0	1
45	0	2	0	70

Table 5c

Pivotstr				
Tenant	Table	Col	Row	Str
22	0	0	0	Arya
22	0	3	0	Apollo
22	0	0	1	Girish
22	0	3	1	St. John
37	1	0	0	Bipin
45	2	0	0	Bhaskar

→ Chunk Table Layout: A third generic structure called chunk table is specified here. Chunk table is like a pivot table execept that it has a set of of data columns of various tables and Col column of pivot table is replaced by a Chunk column. A logical source table dataset will be partioned into dense subsets. These subsets are placed under a group of columns in the chunk table and assigned a chunk id.

The advantage of this method as compared to Pivot table layout is , it reduces the ratio of stored metadata-to actual data and also the overhead for reconstructing the logical source

table.As compared to Universal table layout, overly-wide columns are broken down and supports typing.

Chunk table layout

Table 6a

Account17

	Aid	Name	Hospital	Beds
Row:0 →	22	Arya	Apollo	120
	22	Girish	St.John	1255

Table 6b

Chunkstr / int

Tenant	Table	Col	Row	Int	Str
22	0	0	0	1	Arya
22	0	3	0	120	Apollo
22	0	0	1	2	Girish
22	0	3	1	1255	St.John
37	1	0	0	1	Bipin
45	2	0	0	1	Bhaskar
45	2	0	0	70	,--

→With the above techniques in hand, [5] has proposed a new technique called Chunk Folding. In this technique, the logical source tables are vertically partitioned into chunks that are folded together into different physical multi-tenant tables and joined as needed. In this approach, the most heavily utilized parts of logical schemas of the tenants are mapped into the conventional tables and the remaining parts into the chunk tables that match their structure as closely as possible hence the metadata budget is effectively managed between conventional and chunk tables.

Chunk Folding table layout

Table 7a

AccountRow

TenId	Row	AId	Name
22	0	1	Arya
22	1	2	Girish
37	0	1	Bipin
45	0	1	Bhaskar

Table 7b

ChunkRow

Tenant	Table	Chunk	Row	Int1	Str1
22	0	3	0	120	Apollo
22	0	3	1	1255	St.John
45	2	0	0	70	--

3.2 Implementing Multi-tenant database using native support of RDBMS

The storage models mentioned above to implement multi tenant databases have several advantages like, it reduces cost per tenant, and generic structure poses no limitation on consolidation and extensibility of tenant specific attributes. However, Oliver Schiller and et al., in [6], points out that these approaches also have several drawbacks as it implements most per-tenant operations such as schema extension, backup and recovery in the application itself. The authors also mentions that query re-writer component used in the application conducts a major part of data dictionary management like:

- ➔ Storing data type of a tenant specific attribute.
- ➔ Maintaining the tenant's logical query structure.
- ➔ Enforces isolation of Meta data and application data between tenants.

For these reasons, query re-writer is a complex and critical component that requires a clean design and sophisticated testing.

In order to overcome the above mentioned drawback authors of [6] have suggested a different approach in implementing multitenant database. This approach allows maintaining an application core schema while enabling per tenant schema extension. Also, it is the RDBMS that maintains the data dictionary such as data types of tenant specific attributes instead of application itself maintaining all the metadata information of the database. This approach also facilitates direct access to RDBMS without need of complex query rewriter which relies on native support of multitenancy in RDBMS.

3.3 XML Support to implement multitenant database

Franclin S. Foping et al., in [9] has proposed a hybrid schema sharing technique for multitenant applications. They basically maintain two separate tables. One is a tenant table where in all common data are stored [Table 8a].

Table 8a

Manager ID INT	Username VARCHAR	Full Name VARCHAR	Contact INT	Coun -ty ID INT
1	Ramachan	Ramachandra Nayak	0831456721	2
2	Ashutosh	Ashutosh Agarwal	0841786549	1
3	priya123	Priyanka Choudary	0832567842	3

The other is an extension table [Table 8b] which stores tenant specific data in XML form as shown.

Table 8b

Manager ID INT	Extension XML
1	<Parameters><Hospital name="Apollo"> <Beds>120</Beds> </Hospitalname> </Parameters>

Whenever a tenant throws a query, it is executed by attaching a tenant-id at runtime.
The advantages of storing data in XML form are:

- Querying of tenant specific data will not be hindered as null values are avoided.
- Indexes can be applied on tenant fields and hence queries are optimized on a per tenant basis.

However the drawback of this approach is that XML document cannot be validated against a schema as this document will be generated at runtime.

4 SECURITY

The risks involved in SaaS applications accessing databases are been divided into three categories in [10].

- Managing user identity
- Managing user access
- Managing user credentials.

The same is applicable for SaaS applications handling multitenant databases as well. Hence there is a need to address these issues in multitenant environment.

In an article named "Access Control in Multitenant applications with visual guard" [13], author Novalys talk about securing multitenant application with access control mechanisms like user authentication and user permissions. According to the author, any multitenant application need to meet two criteria -> protecting data from other tenants -> delegate administrative privileges which can be achieved using access control systems like Visual Guard which works with various application architectures.

In [14] author Mr.Ramkumar suggests that in a multitenant environment the older paradigm of roles and page level access control could be replaced by new paradigm of privilege based access controls. i.e., instead of hard coding access control policies of the product into the roles, a user is checked for the privilege for doing any action and user/roles privileges are resolved during runtime but not hard coded at design time.

However the author also mentions that not only the roles/privileges could be mapped during runtime but also the data scopes should be configurable by an end customer and mapped with each role/privilege mapping. This control configuration can then be stored tenant wise. During runtime appropriate tenant specific access control setting can be looked up and decide to allow or disallow a particular action in the application.

Thus security in multitenancy opens up several issues which could be understood researched and addressed.

5 CONCLUSION

Multitenancy is a promising paradigm which enables sharing of a single service instance among multiple tenants and also leverages benefits for service provider. However in this process of sharing, several open issues needs to be addressed related to resource sharing, scalability and security. This pro posed survey provides researchers the idea on current mul titenant systems, hype and challenges.

ACKNOWLEDGMENT

The authors would like to acknowledge and thank Technical Education Quality Improvement Program [TEQIP] Phase 2, BMS College of Engineering.

REFERENCES

[1] Bhaskar Prasad Rimal,Enumi Choi, Ian Lumb, "A Taxanomy and Survey of Cloud Computing System", Fifth International Joint Conference on INC, IMS and IDC 2009.

[2] Software as a Service, http://en.wikipedia.org/wiki/-Software_as_a_service.

[3] Craig D Weissman, Steve Bobrowaki , "The Design of the Force.com Multitenant Internet Application Development Platform", SIGMOD'09, June 29 - July 2, 2009, Providence, Rhode Island, USA., ACM 978-1-60558-551-2/09/06, 2009.

[4] Sungjoo Kang, Sungwon Kang, Sungjin Hur, "A Design of the Conceptual Architecture for a Multitenant SaaS Application Platform", First ACIS/JNU International Conference on Computers, Networks, Systems, and Industrial Engineering 978-0-7695-4417-5/11, IEEE 2011.

[5] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, Jan Rittinger, "Multitenant Databases for Software as a Service-Schema Mapping Technique", SIGMOD'08, June 9–12, 2008, ACM 978-1-60558-102-6/08/06 2008.

[6] Oliver Schiller, Benjamin Schiller, Andreas Brodt, Bernhard Mitschang, "Native Support of Multi-tenancy in RDBMS for Software as a Service", EDBT 2011, March 22–24, 2011, Uppsala, Sweden, ACM 978-1-4503-0528-0/11/0003, 2011.

[7] Dean Jacobs, Stefan Aulbach, "Ruminations on multi-tenant databases", In Datenbanksysteme in Business, Technologie und Web, pp. 514-521,2007, ISBN: 978-3-88579-197-3Digital Object Identifier 10.1.1.140.6429

[8] Frederick Chong, Gianpaolo Carraro, Roger Wolter, "Multitenant Data Architec-

ture" Microsoft Corporation, http://msdn.microsoft.com/en-us/library/aa479086.aspx, 2006.
[9] Franclin S. Foping, Ioannis M. Dokas, John Feehan, Syed Imran, "A New Hybrid Schema- Sharing Technique for Multitenant Applications"
[10] Cisco SaaS access control http://www.cisco.com/c/en/us/products/collateral/security/anyconnect-secure-mobility-client/white_paper_c11-596141.html
[11] Securing Multitenancy and Cloud Computing ,White Paper, Juniper Network, 2000381-002-eN Mar 2012
[12] Steve Bobrowski, "Optimal Multitenant Designs for Cloud Apps", 4th International Conference on Cloud Computing, 978-0-7695-4460-1/11, ISBN: 978-0-7695-4460-1 Digital Object Identifier 10.1109/CLOUD.2011.98 IEEE 2011.
[13] Article by Novalys, Access Control in Multi-Tenant Applications with Visual Guard, 2011.
[14] Article by Rajkumar R.S.,Access Control in Multi-tenant Applications , 2012.
[15] Cor-Paul Bezemer, Andy Zaidman, "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?", Report TUD-SERG-2010-031, Delft University of Technology Software Engineering Research Group Technical Report Series 2010.
[16] Peter Mell, Timothy Grance, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, The NIST definition of cloud computing, September 2011.
[17] Sanjeev Pippal,Vishnu Sharma, Shakti Mishra, D.S.Kushwaha, "An Efficient Schema Shared Approach for Cloud based Multitenant Database with Authentication and Authorization Framework", International Conference on P2P,Parallel,Grid,Cloud and Internet Computing, Barcelona, 26-28 Oct. 2011 ,pp 213 – 218, Print ISBN: 978-1-4577-1448-1, Digital Object Identifier : 10.1109/3PGCIC.2011.39
[18] Anthony T.Velte, Toby J.Velte, Robart Eisenpeter, "Cloud Computing: A Practical Approach", Tata McGraw-Hill Publishers, 1st Edition, 2009, ISBN: 0071626948
[19] Cor-Paul Bezemer, Andy Zaidman, "Challenges of Reengineering into Multitenant SaaS Applications" Report TUD-SERG-2010-012, Delft University of Technology Software Engineering Research Group ,Technical Report Series 2010
[20] "Secure Multi-Tenancy for Cloud Architecture with NetApp, Cisco, and VMware", http://www.netapp.com/in/technology/secure-multi-tenancy.html.
[21] Jose M.Alcaraz Calero, Nigel Edwards, Johannes Kirschnick, Lawrence Wilclock, Mike Wray, "Towards a Multitenancy Authorization System for Cloud Services", 1540-7993/10, IEEE 2010.
[22] http://www.wolfframeworks.com/benefits.asp