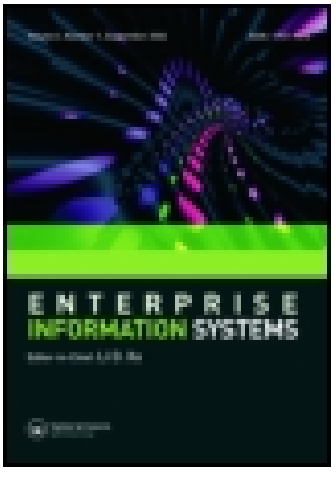


Enterprise Information Systems
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/teis20

Extending multi-tenant architectures: a database model for a multi-target support in SaaS applications

Antonio Rico[a], Manuel Noguera[a], José Luis Garrido[a], Kawtar Benghazi[a] & Joseph Barjis[b]

[a] Department of Software Engineering, University of Granada, E.T.S.I.I.T., c/ Saucedo Aranda s/n, 18071 Granada, Spain

[b] Department of Systems Engineering, Delft University of Technology, Jaffalaan 5, BX 2628 Delft, The Netherlands

Published online: 15 Sep 2014.

To cite this article: Antonio Rico, Manuel Noguera, José Luis Garrido, Kawtar Benghazi & Joseph Barjis (2014): Extending multi-tenant architectures: a database model for a multi-target support in SaaS applications, Enterprise Information Systems, DOI: 10.1080/17517575.2014.947636

To link to this article: http://dx.doi.org/10.1080/17517575.2014.947636

Enterprise Information Systems, 2014
http://dx.doi.org/10.1080/17517575.2014.947636

Taylor & Francis
Taylor & Francis Group

Extending multi-tenant architectures: a database model for a multi-target support in SaaS applications

Antonio Rico[a]*, Manuel Noguera[a], José Luis Garrido[a], Kawtar Benghazi[a] and Joseph Barjis[b]

[a]*Department of Software Engineering, University of Granada, E.T.S.I.I.T., c/ Saucedo Aranda s/n, 18071 Granada, Spain;* [b]*Department of Systems Engineering, Delft University of Technology, Jaffalaan 5, BX 2628 Delft, The Netherlands*

(*Received 6 October 2013; accepted 20 July 2014*)

Multi-tenant architectures (MTAs) are considered a cornerstone in the success of Software as a Service as a new application distribution formula. Multi-tenancy allows multiple customers (i.e. tenants) to be consolidated into the same operational system. This way, tenants run and share the same application instance as well as costs, which are significantly reduced. Functional needs vary from one tenant to another; either companies from different sectors run different types of applications or, although deploying the same functionality, they do differ in the extent of their complexity. In any case, MTA leaves one major concern regarding the companies' data, their privacy and security, which requires special attention to the data layer. In this article, we propose an extended data model that enhances traditional MTAs in respect of this concern. This extension – called multi-target – allows MT applications to host, manage and serve multiple functionalities within the same multi-tenant (MT) environment. The practical deployment of this approach will allow SaaS vendors to target multiple markets or address different levels of functional complexity and yet commercialise just one single MT application. The applicability of the approach is demonstrated via a case study of a real multi-tenancy multi-target (MT^2) implementation, called *Globalgest.*

Keywords: enterprise information systems; multi-tenancy; multi-target; software architecture; database design; cloud computing; Software as a Service

1. Introduction

Cloud computing (CC) has become one of the top priorities of many chief information officers and is rapidly moving from early adopters to mainstream organisations (Xu 2012). Software as a Service (SaaS) is considered one of the three main service models in CC (Mell and Grance 2011). In SaaS, applications are no longer a product but a service. Vendors develop solutions that are offered through Internet as a service and customers (now *tenants*) pay for its use on a regular basis. Conditions for this service are agreed in the service-level agreements (SLAs) through a formal contract (Prabowo, Janssen, and Barjis 2012; Kaplan 2007). Success of the everything-as-a-service (XaaS (Xu 2012; Ambrust et al. 2009; Silva and Lucrédio 2012; Goedert 2012) formula facilitated CC to evolve to new paradigms in specific domains. That is the case of *Cloud Manufacturing* (CMfg) (Xu 2012) where resources related to different stages in the product life cycle are encapsulated and served on demand, following optimal composition algorithms (Tao et al. 2013).

*Corresponding author. Email: antoniorico@ugr.es

Enterprise information systems (EIS) have emerged as a promising tool for integrating and extending business processes (Xu 2011a). Over the past decade, more and more enterprises have adopted EIS for running the company (Li et al. 2012). In this SaaS model, companies do not need to invest in either IT infrastructure, acquisition or development of EIS; besides, operational costs are distributed over all tenants, and hence, expenses are significantly reduced. In the very core of SaaS, success lays an innovative architecture called multi-tenant architecture (MTA).

Multi-tenancy refers to the ability that one system has in order to serve multiple tenants with just one software instance. SaaS providers can now leverage economies of scale by serving many companies with just one application. This sharing can be achieved at system level by means of other technologies such virtualisation (Li et al. 2008; Shroff 2010). However, virtual machines set logical boundaries between tenants' resources; a separation that affects system efficiency (Bezemer and Zaidman 2010a) and eventually incurs additional costs. This pattern is not able to cater for the demand (Li et al. 2008) and imposes a much lower limit on the number of tenants allocated (Bezemer and Zaidman 2010a). In this article, we discuss the application level where multi-tenancy is implemented through software architecture. With this pattern, server usage is more efficient and costs are lowered. In contrast, demanding tenants may consume many resources compromising the system performance; this situation is solved by balancing the system through farms and/or applying other methods for performance isolation (Li et al. 2008).

The market of multi-tenant (MT) applications for enterprises is vast, from *Enterprise Resource Planning* (ERP), *Customer Relationship Management* (CRM) to *Supply Chain Management* (SCM) or *Content Management System* (CMS). Companies willing to adopt the cloud will find a wide variety of SaaS solutions satisfying their functional needs and requirements. These functional needs will likely differ from one company to another, either because of the type of application, nature of the industry or simply because they demand distinct levels of complexity.

Traditionally, as on-premises solutions do, on-demand MT applications deploy a single functionality. Finding an *all-in-one* service covering all our necessities is a challenging task and quite likely impossible in most cases. Consequently, companies will have to subscribe to as many applications as their business activities need to be supported. On the other hand, providers target just one SaaS niche per application, depending on the functionality deployed.

Independent of the application's functional area, there are common components of the architecture that could be shared across all functionalities (Rico et al. 2012; Rico Ortega, Noguera, and Garrido 2013). In current practice, while the login process or database-connection libraries might be identical in a CRM or CMS, they are often replicated for different implementations.

The proposed approach, called multi-tenancy multi-target (MT^2), unifies these multiple implementations in one by reusing the common components of the architectures. MT^2 applications can host several functionalities (CRM, ERP, CMS, etc.) and serve them selectively as a subset that depends on tenant's contract. Thanks to this unification, customers benefit from new price reductions and vendors are able to target multiple markets with just one application instance.

The MT^2 idea and approach have been quite extensively presented in the literature. In Rico et al. (2012), authors introduce the MT^2 approach by explaining its foundations and providing a general idea of the architecture. Subsequently, MT^2 architecture (MT^2A) tiers and components are more profoundly detailed and explained in Rico Ortega, Noguera, and Garrido (2013); benefits in terms of agility and support to rapid development and deployment are studied in Rico et al. (2013).

In shared environments such as SaaS, data privacy and security are key; confidence in the system relies on them, and any breach would undermine the use of the system. Therefore, the data layer of traditional MTAs has attracted serious attention in the literature. In MT^2, new aspects must be considered, and in this article, our contribution is focused on the study and extension of MT^2A in regard to the data layer. As extension to traditional MT models, in this article we consider *pure multi-tenancy* (Bezemer and Zaidman 2010a) by using the *shared schemas* (Chong et al. 2006; Jacobs and Aulbach 2007) approach as base model.

The rest of the article is organised as follows: Section 2 summarises the concept of multi-tenancy and the general model of its architecture. Section 3 reviews the state of art for MT databases. Sections 4 and 5 explain an approach of a database model that extends traditional MT databases to MT^2 so as to support multiple functionalities. Section 6 explains a case study and presents a real MT^2 implementation called Globalgest. Section 7 discusses the approach and reveals some shortcomings identified. The final section summarises conclusions and future work.

2. Multi-tenant architectures

Software architecture describes a set of system components as well as their topological relations (Bass, Clements, and Kazman 2003). A significant contribution to software architecture is the definition of patterns that can be used as guides and blueprint when designing, for example, an EIS (Niu, Xu, and Bi 2013).

Multi-tenancy is an architectural pattern for SaaS applications that permits several customers (tenants) to share the same instance of a software (Bezemer and Zaidman 2010a). A tenant is an organisational unit that pays for the use of an SaaS application on a regular basis according to a certain subscription contract. One tenant might consist of many end-users; therefore, MT applications might be also multi-users; we call *tenancy* the set of users of one tenant that run the same customised version of the application instance. The number of instances running on a MT environment might be more than one, thus resulting in a *multi-tenancy farm* (Jacobs and Aulbach 2007). This situation could occur not only because of performance issues as some tenants might get greedy on computer resources but also because of country legislations stating the obligation to store data within country borders (Bezemer and Zaidman 2010b; Neves et al. 2011). A major consideration in SaaS is effective integration with other applications (Xu 2012). MT systems should consider a mechanism in order to integrate services from different vendors. This situation may occur not only because of lack of features but also because of the very heterogeneous nature of organisations (like networking enterprises (Li et al. 2010)). Basically, service incorporation is achieved by means of *Cloud Integration Agents* (CIAs) present on the different systems that communicate with each other through a *Cloud Integration Platform* (CIP). The CIP deals with key aspects such as contract license, service access path or business information path (Li, Wang, Li, Cao, et al. 2013). In this regard, there are recent studies that further study service convergence and multi-cloud integration (Li et al. 2010; Li, Wang, Li, Li, et al. 2013; Li, Wang, Li, Cao, et al. 2013).

The MT model is considered an essential characteristic for SaaS and CC (Mell and Grance 2011; Qaisar 2012). Salesforce.com, a popular SaaS provider, states (Coffee 2007): 'hosting models that do not offer the leverage of multi-tenancy do not belong in the same discussion as the value proposition implied by the term SaaS'. Chong (Chong and Carraro 2006) believes that three attributes are to be considered in a good SaaS application architecture: scalability, configurability and MT efficiency.

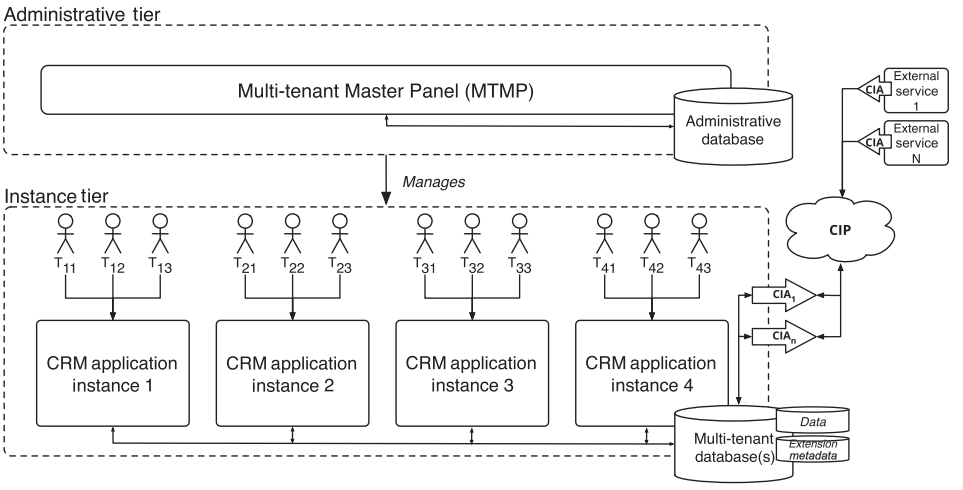


Figure 1. Tiers in MTAs: administrative and instance level.

In the study of MTA models, two tiers (or levels) can be identified: administrative and instance. Figure 1 shows an example of these two tiers in the architecture of a traditional MT System:

- **Administrative tier**: Administrative framework responsible for managing and controlling the MT environment as a system. Data in this level are stored in the administrative database. The administrative level, among others, defines the characteristics of the subscription for each tenant.
- **Instance tier**: Applications that tenants share and execute. The functionality deployed is common to all tenancies. Tenants store data into a MT database. Tenants can individually extend (customise) the common database model for by means of extension metadata.

The administrative level controls and manages instance level where the tenants run the service contracted. In this case, the system deploys CRM functionality that is shared among 12 tenants within a farm of four application instances. CIA components open the door to outside services. Data from different clouds can be stored/served in/out the MT database after communicating and negotiating through CIP. As stated before, administrative tier and component-based design has been discussed and introduced in Rico Ortega, Noguera, and Garrido (2013).

3. Multi-tenant databases: state of art

In shared scenario like MT applications, databases are key since companies consider system customisation, security and data privacy are major concerns for SaaS adoption by companies (Sultan 2011). At data level, multi-tenancy is achieved by transforming the individual logical tables of one tenant into the communal physical ones. This conversion is made by means of schema-mapping techniques and the use of metadata.

Figure 2 depicts the instance level of the architecture – greyed out components are related to the data level. The lower level tiers perform changes dictated by business layer in both MT database and file system. The *Query Processor* maps the logical tables into the physical ones. Intermediate layers such as presentation or SOA services communicate

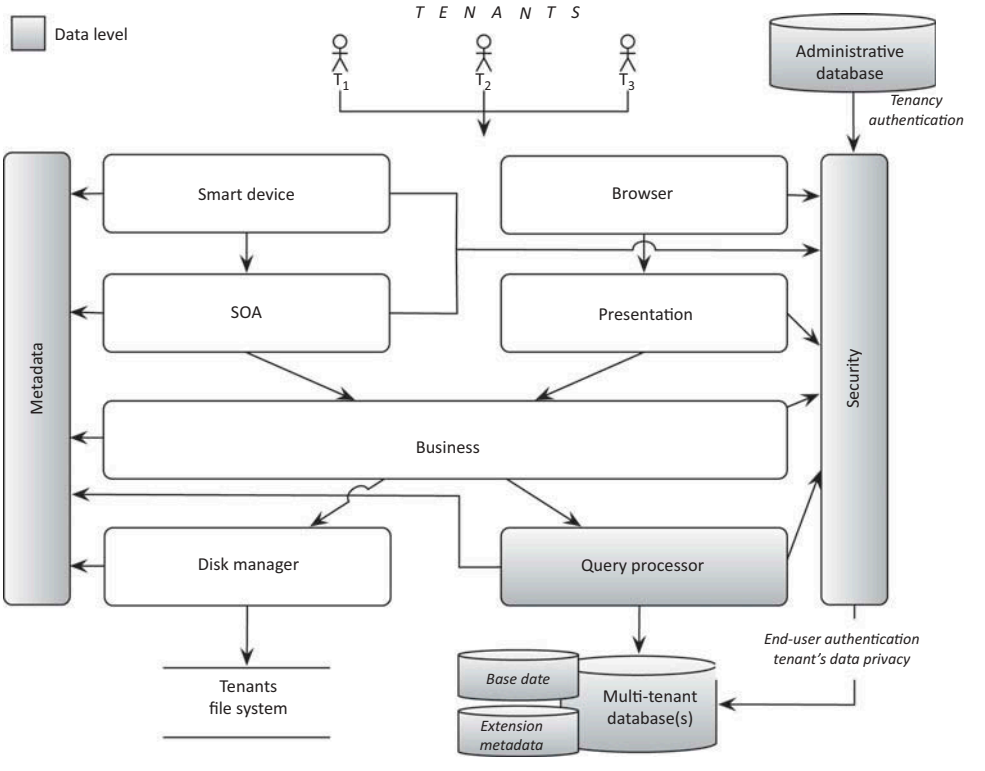


Figure 2. Multi-tenant architecture detailed: instance tier [based on Chong and Carraro (2006)].

with browser and smart devices, respectively, to produce end-users output. Metadata are responsible for data extensibility for system customisation and security services maintain data privacy.

There are many authors who have studied several approaches for implementing MT databases. Though with different denominations, literature agrees to categorise them depending on the isolation degree on tenants' data. While *isolated* approaches store tenants' data separately, a *shared* approach will save tenants' data into the same location (Figure 3). According to Chong et al. (2006), separation is not binary but continuum; approaches could provide a level of isolation positioned between the two extremes.

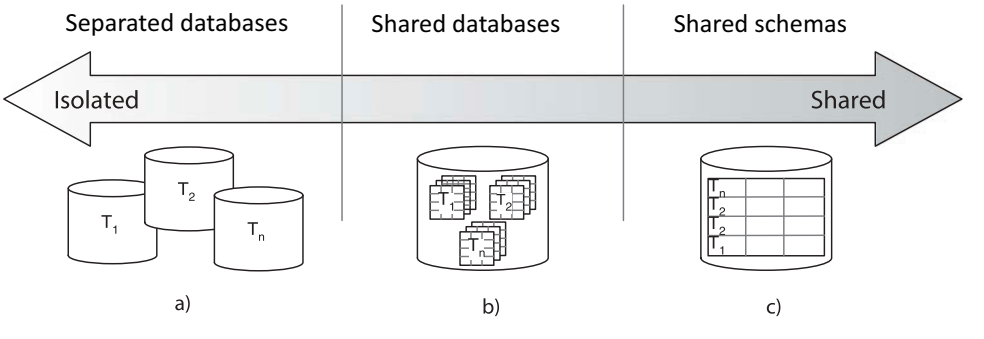


Figure 3. Continuum separation of data isolation (Chong et al. 2006). Base model approaches.

Along with security and privacy issues, extension of the database model is also crucial; customers should be able to integrate tenants-specific customisations (Liao, Chen, and Chen 2013). According to Chong and Carraro (2006), the more isolated the data are, the easier to customise but the more expensive are the hardware and overall maintenance.

MT systems are also applications themselves that need to be controlled and managed. Therefore, when implementing an MT application, three models must be considered in the database design:

- An *administrative* model for vendors' management.
- A *base* model defining a mutual structure for all tenants.
- An *extension* model allowing tenants to individually customise the application.

3.1. Administrative model

SaaS MT applications are served to organisations on a subscription basis. In fact, there is a common perception to consider the future of computing as the fifth utility (Zhang, Cheng, and Boutaba 2010; Buyya et al. 2009; Carlin and Curran 2012; Fox et al. 2009). After water, gas, electricity and telephony, we should be able to access computing on demand and pay regularly for this consumption. The same way, these utility companies make use of information systems to manage and control the service; SaaS MT providers need an administrative level (Silva and Lucrédio 2012; Rico Ortega, Noguera, and Garrido 2013; Jacobs and Aulbach 2007) in the application containing tables for:

- Tenants contract management
- System configuration
- Performance monitoring
- Balance control (just for multi-tenancy farm)

This model does not involve applying schema-mapping techniques for achieving multi-tenancy (although could be considered for resellers). Hence, we will not go into deep discussion here.

3.2. Base model

Regarding the isolation degree of the data layer, three approaches are studied:

- *Separate databases* (Chong et al. 2006) [*Shared Machine* (Jacobs and Aulbach 2007)]: Information of the tenants is stored in different databases. High degree of isolation (Figure 3a).
- *Shared database* (Chong et al. 2006; Jacobs and Aulbach 2007): Tenants use the same database, but information is stored in separated schemas. Medium degree of isolation (Figure 3b).
- *Shared schemas* (Chong et al. 2006) [*Shared Tables* (Jacobs and Aulbach 2007)]: Tenants share database instance and schemas. Each record of common tables' data must contain a reference to the tenancy proprietary. Lowest degree of isolation (Figure 3c).

Some authors consider multi-tenancy pure when using low degrees of isolation (like in the shared schemas approach); other variations where reutilisation of resources is not

maximised are considered as semi-multitenant (Bezemer and Zaidman 2010a). In Wang et al. (2008), the authors conduct a study for performance evaluation; better results are achieved as we approach the isolated side of the spectrum. However, since tenants get their own resources, these can be under-used; a scenario that affects system efficiency increases costs and limits the number or tenants allocated.

3.3. Extension model

The extension model permits tenants to personalise the database model to its individual needs. At data layer level, extending isolated approaches is easy; each tenant can modify its scheme and changes made will only affect to its own tenancy. However, this model increases infrastructure and maintenance expenses.

In shared environments like *shared schemas,* complexity for customisation is much higher. Custom logical tables of tenants must be mapped into a common physical schema valid for all tenancies. This situation reduces costs of maintenance and infrastructure but on the other hand incurs in a much higher effort of design and development. In the following lines, we summarise extensibility patterns for mapping logical into physical tables within system with *shared schemas* as base model. *Extension Tables* (Aulbach et al. 2008) model is left behind because it is considered a hybrid layout that isolates customisation tables and does not fit in pure multi-tenancy.

Universal table (Aulbach et al. 2008): This approach – adopted by Salesforce.com (Weissman and Bobrowski 2009) – consists of storing data from all tables and tenants within the same generic structure. The table is conformed by a large number of flexible VARCHAR fields together with the *tenant_id* and *table_id* to keep track of the record's logical precedence. Although with NULL cells, this method requires very wide rows and might compromise system performance.

Pivot table (Aulbach et al. 2008): Generic structure where every column of the logical tables is mapped into a physical row. Pivot tables must track *column_id* and *row_id* besides using a flexible type as VARCHAR for the data field. Although eliminates the issues with NULL values in the universal tables, it incurs in a large number of rows. In order to reduce overhead and support indexing, pivot tables could be divided into two.

Sparse columns (Aulbach et al. 2009): Extension fields are implemented by a sparse set of fields storing name–value pairs of the tenant. This approach was originally developed by Microsoft SQL server 2008 in order to solve NULL values performance issues.

Chunk table (Aulbach et al. 2008): Another generic structure similar to pivot tables but more effective in certain situations. Data are divided into chunks and *column_id* is replaced by *chunk_id* column.

Chunk folding (Aulbach et al. 2008): In this approach, common base data are separated from the extension information that is stored in chunk tables.

XML: Customised fields are stored as an XML within an extra field of the physical table. *PureXML* (Saracca, Chamberlin, and Ahuja 2006) is a commercial implementation of IBM for this approach. In Foping et al. (2009), the authors propose a combination of this approach and the extension table layout. Use of XML or any other mark-up language (JSON) can be combined with others approaches.

Hbase – Google BigTable (Aulbach et al. 2009; Chang et al. 2008): Hbase is the Apache open source version for Google BigTable. Personalised data of the tenants are stored in the columns of a column family.

Preallocated fields: In Chong et al. (2006), Chong applies the universal table approach preallocating extension fields on each base table. Definitions for extension fields

are stored in metadata tables. The number of extra fields allocated in metadata tables can either limit tenant's extension capacity or be source of overhead.

Name–value pairs (Chong et al. 2006): Prevent limitations of the *preallocated fields* approach by including one extension field definition per row to the detriment of simplicity of database functions.

Elastic extension tables (Yaish, Goyal, and Feuerlicht 2011): As well as extending common base tables with new columns, tenants are allowed to define and map its own virtual database schema within a set of physical tables. This includes definition of new tables and columns.

4. Multi-tenant multi-target architectures

In traditional SaaS MT systems, each MT application usually deploys a single functionality that is pooled among tenants with similar functional needs. Vendors can only focus on limited business domains (Li, Wang, Li, Cao, et al. 2013) and cannot satisfy the full functional requirements of an enterprise (Li, Wang, Li, Li, et al. 2013). In this regard, we could call current MT applications mono-target because they target a single area in the spectrum of potential clients (Rico et al. 2013). Mono-target systems deploy a single functionality or are aimed to serve one single specific line-of-business (LOB) such as insurances, health care, real estate.

In the mono-target approach, applications are shared among tenants with common functional needs. Due to this mono-functional scenario, clients will have to subscribe to as many services as applications needed. Moreover, many architecture components like system authentication or file system access libraries that could be shared among different applications (CRM, ERP or CMS) are however replicated across mono-target MT systems. In Figure 4a, not only tenant 2 has to subscribe to three different applications incrementing costs (among others disadvantages (Rico et al. 2012, 2013)) but also programmers need to clone common development components over different implantations, which increases efforts and hence adds to time to market.

Rationality is simple: Why replicate identical elements across different software applications? If applications could be unified somehow, would it be possible to reuse those elements?

Multi-tenancy multi-target is an approach to extend MTAs so that multiple functionalities can be served with the same software instance. This way, tenants needing functionalities of disparate nature or simply with dissimilar level of complexity will be able to unify services using one single application.

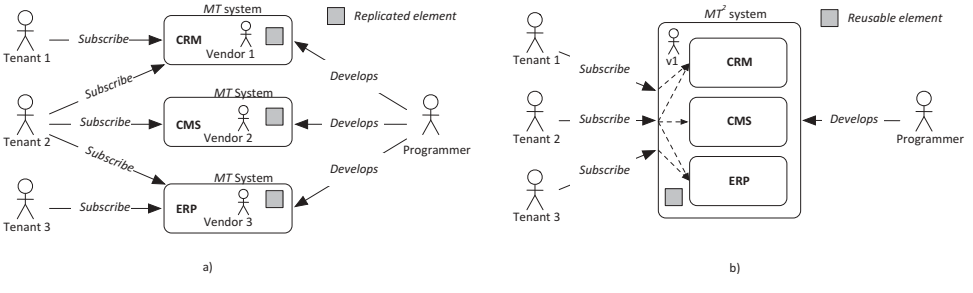


Figure 4. (a) Multi-tenancy mono-target scenario and (b) multi-tenancy multi-target scenario.

The set of functionalities deployed in a MT^2 system is called *functional portfolio*. The number of functionalities in the portfolio may differ depending on vendor. MT^2 systems seek for scalable not only at tenant level but also at functional level. New MT^2 systems may deploy just a few features but can increase portfolio across the time. Former MT^2 systems are supposed to have larger functional portfolios, since new functionalities are added on customers' demands and remain on the portfolio, unless outdated. In MT^2, vendors can easily expand to other LOBs with fewer risks due to the native functional scalability of the system. Software companies broaden the spectrum of potential clients in addition to reducing the general costs of the software maintenance and infrastructure. Likewise, prices are lowered to clients, who also reduce complexity by merging their software subscriptions.

As depicted in Figure 4b, in a MT^2 scenario tenant 2 unifies its subscriptions to a single application served by a single vendor (v1), who can now sell to the CMS and ERP markets. From the developers' perspective, programmers do no longer need to replicate, but to reuse common components. MT^2 eases programming effort and supports agility (Rico et al. 2013) by leveraging reusability of common components.

As for the concern of outside service integration, we must consider another scenario where the functionality contracted is served by an external vendor. In Figure 5, the MT^2 functional portfolio is shorter and just presents CMS and ERP. In order to satisfy the CRM functional needs, CIAs and the CIP collaborate to integrate this external service provided by vendor 2 into the MT^2. This situation involves benefits for vendors and clients: vendors are able to increment business due to externalisation, whereas clients have their functional needs satisfied. Besides, CIAs of the MT^2 systems are part of the reusable components and do not need to be replicated.

Figure 6 illustrates the general model of the MT^2A. The administrative level (now represented by the MT^2MP) has to deal with a different kind of multi-service subscription and needs to specify what functionalities from the portfolio are to be served. In turn, the instance tier will selectively deploy functionalities on demand according to tenant's subscription contract. As for the integration, MT^2 goes a step further of traditional MT by enabling the incorporation of full functionalities instead of simple features or data integration. As we see on the figure, instance 4 of the systems outsources the CRM functionality for T_{41} and T_{42}. CIA components in multi-target systems work not only at data level but also at functional by serving entire functionalities from other providers.

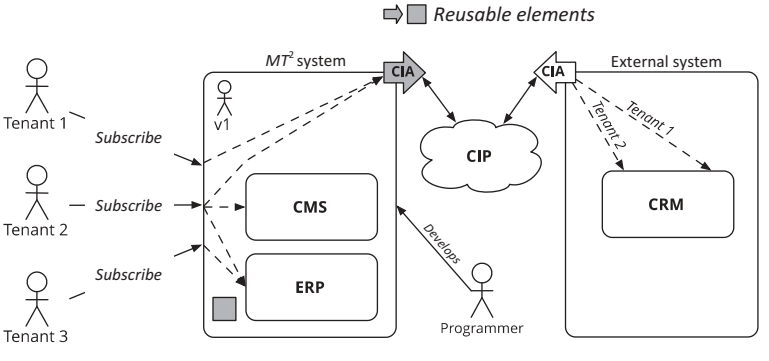


Figure 5. MT^2 scenario with CRM external service integration.

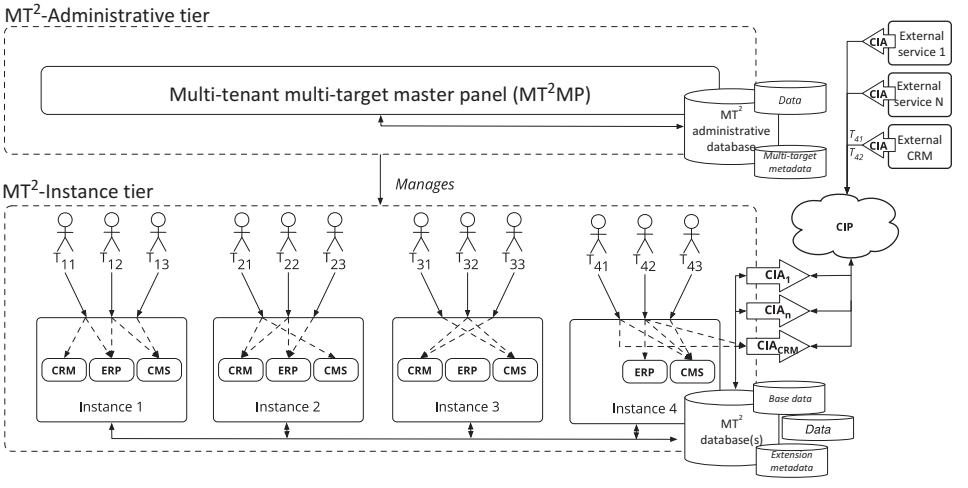


Figure 6. Tiers in MT²As: Administrative and instance level.

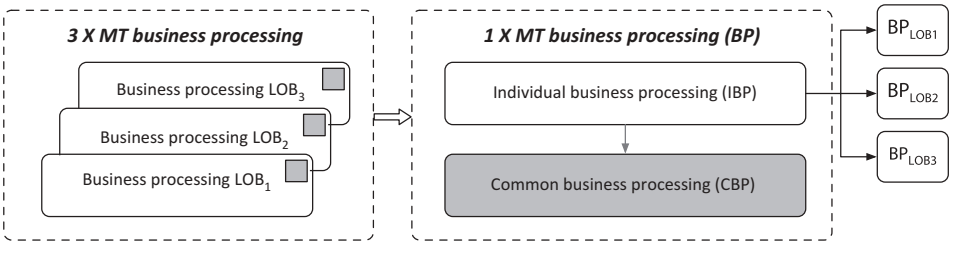


Figure 7. MT² architecture unifies MT business processing layer.

Basically, the main idea behind this approach is reusability. Common components from the lower level tiers of the MTA are no longer replicated but reused, and thus, different functionalities mean no longer different implementations. To this end, the previous business-processing layer is divided in two; as seen in Figure 7, the business logic specific for individual LOBs is deployed selectively in the *individual processing business* (IBP) layer, whereas common logic and components are consolidated into the common business-processing (CBP) layer. In MT², there is one software instance that encompasses all functionalities and deploys them selectively depending on each tenant's contract.

5. Multi-tenant multi-target databases

In MT² environments, multiple functionalities are served to multiple organisations with many end-users with just a single MT application. To this end, several further aspects must be considered in comparison with mono-target databases.

Figure 8 illustrates the MT² instance architecture with the extended components drawn in bold line for distinction; similarly, components filled in grey are related to the data level.

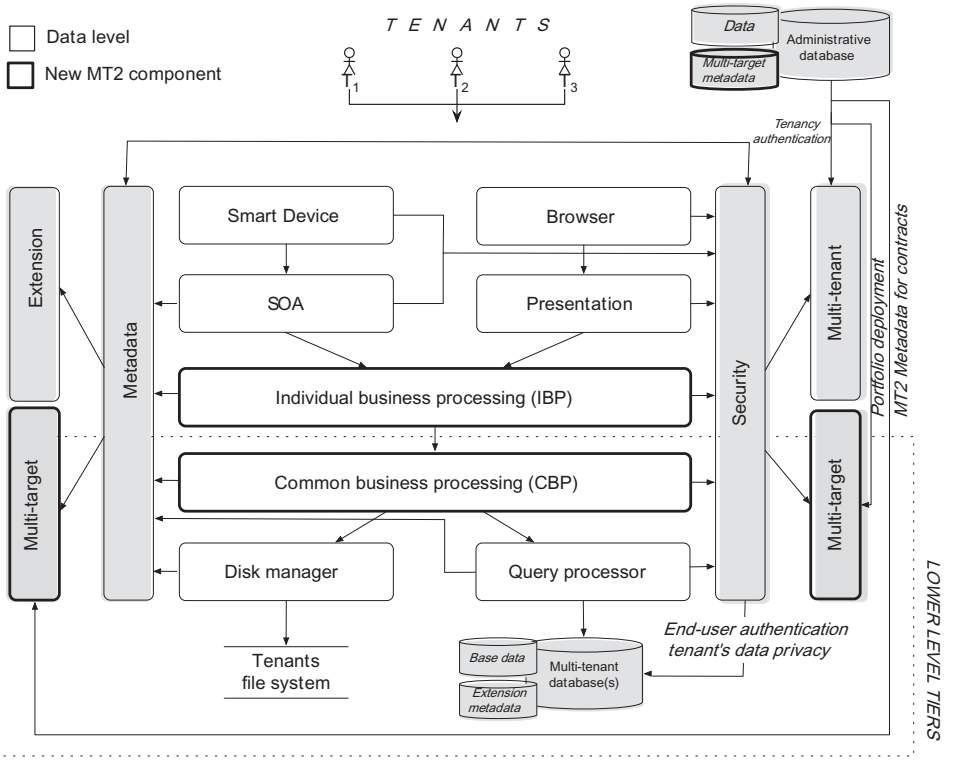


Figure 8. MT^2 Architecture detailed: instance tier.

In this extension, major modifications are focused in the administrative level; the administrative database must include new multi-target metadata to control what functionalities to deploy for each tenant as not all functionalities are to be deployed at instance level. In this article, a version of the name–value pairs approach will be applied for the extension metadata.

5.1. Administrative model

Figure 9 represents the administrative database model in MT^2. The *administrative tenants table* (ATT) stores tenants' information and has no difference with respect to mono-target models. The *tenant_id* field will be used as foreign key across all tenants tables (TTs).

Functionalities are located in the *administrative functional table* (AFT). This table represents the functional portfolio offered by the MT^2 system. The number of functionalities in the portfolio may differ depending on vendor and can increase over time.

Administrative table of application profiles (ATAP) stores functional profiles and their definitions. Different functionalities might share the same profile; for example, SMS and newsletter could be categorised with the CRM profile, as its major aim is gathering leads and sales opportunities. On the other hand, billing could be catalogued in the ERP profile. Besides, having statistics of this segmentation might be worthwhile and of good help for providers; vendors could study the system usage and subscriptions not only at functional level but also at profile level.

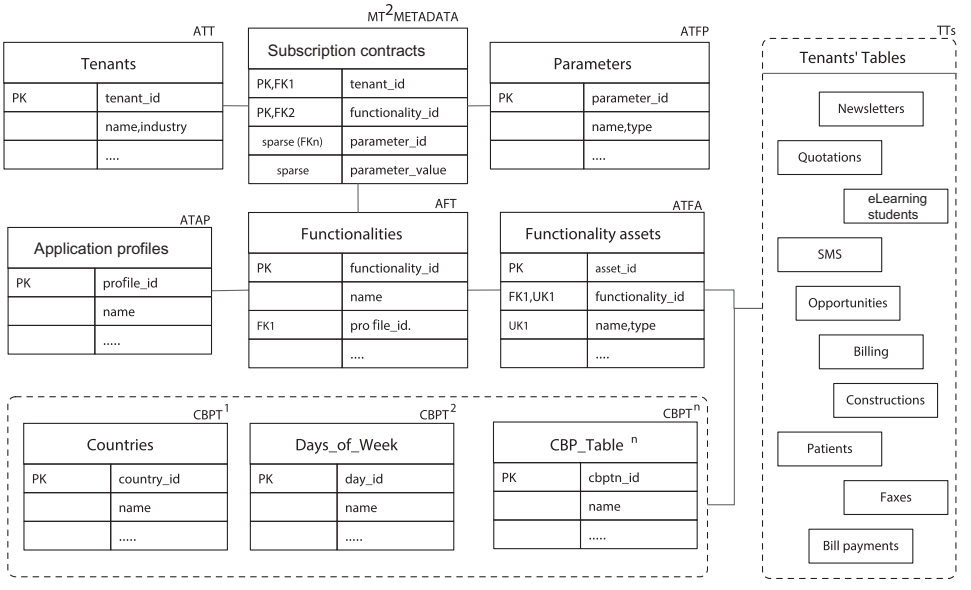


Figure 9. Administrative model in MT^2.

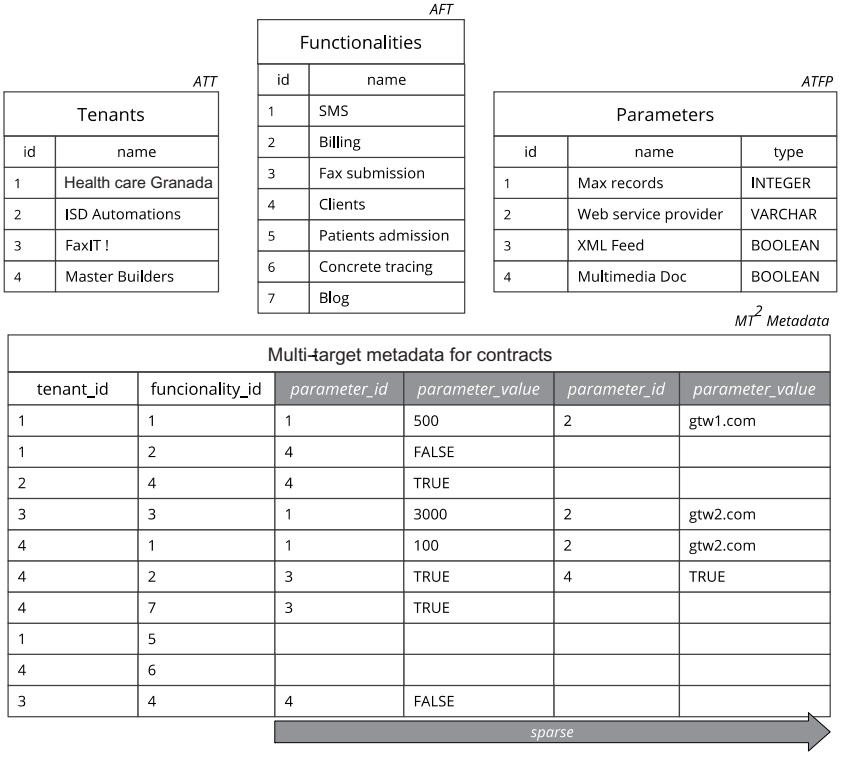
A single functionality might comprise many base tables to store tenants' data (tables or relations). The *administrative table of functionality assets* (ATFA) represents a relation of TTs and AFT (associations can be extended to any software asset but we will focus on TTs). The relation must be used in security since tenants not subscribed to a specific functionality should not be capable to access to the related tables.

MT^2 metadata link tenants' accounts to functionalities controlling not only subscriptions but also contractual features of this relation. In Figure 9, the *administrative table of functional parameters* (ATFP) contains those parameters susceptible to be part of the contract. The ATFP primary key will be later referenced as a foreign key in the MT^2 metadata so as to instantiate the tenant-specific value for this parameter.

Every subscription to a functionality has its own conditions, and these are reflected on the multi-target metadata for each tenant. For instance, if a tenant wants to subscribe to SMS functionality, at least we should set the number of text messages contracted; setting this parameter in other functionalities, such as *client management*, does not make sense. This scenario requires considering a number of undetermined parameters per functionality in the contract; to this end, we will make use of the sparse column approach.

Figure 10 illustrates a MT^2 system with a portfolio of seven functionalities and four possible parameters. The ATT contains five organisations subscribed to distinct functionalities in different terms. The MT^2 metadata table stores the functional contract that is logically summarised on Figure 11.

As stated before, component reutilisation is the main cause of the MT^2 approach. At database level, there are many tables that are reusable among tenants, regardless of the type of business or functional aim (days of the week, months, countries, currencies, etc.). These *common business tables* are stored in the administrative level of the architecture and linked as foreign keys in the physical tables.



ATT

Tenants	
id	name
1	Health care Granada
2	ISD Automations
3	FaxIT !
4	Master Builders

AFT

Functionalities	
id	name
1	SMS
2	Billing
3	Fax submission
4	Clients
5	Patients admission
6	Concrete tracing
7	Blog

ATFP

Parameters		
id	name	type
1	Max records	INTEGER
2	Web service provider	VARCHAR
3	XML Feed	BOOLEAN
4	Multimedia Doc	BOOLEAN

MT^2 Metadata

Multi-target metadata for contracts					
tenant_id	funcionality_id	*parameter_id*	*parameter_value*	*parameter_id*	*parameter_value*
1	1	1	500	2	gtw1.com
1	2	4	FALSE		
2	4	4	TRUE		
3	3	1	3000	2	gtw2.com
4	1	1	100	2	gtw2.com
4	2	3	TRUE	4	TRUE
4	7	3	TRUE		
1	5				
4	6				
3	4	4	FALSE		

sparse

Figure 10. Sample of tenants' subscriptions table using sparse columns.

Logical subcriptions	
Tenant	Functionalities and terms
Health care Granada	-*SMS* : Max 500 using gtw1.com -*Billing* : No multimedia support -*Patients admission*
ISD Automations	-*Clients* : Multimedia support
FaxIT!	-*Fax submission* : Max 3000 using gtw2.com -*Clients* : No Multimedia support
Master Builders	-*SMS* : Max 100 using gtw2.com -*Billing* : No multimedia support, XML export -Concrete tracing -*Blog* : XML export

Figure 11. Logical representations of tenants' contracts (MT^2 metadata).

5.2. Base model

In the shared schemas approach, the *tenant_id* must be stored as extra field. In contrast to mono-target environments, in MT^2 not all tenants are allowed to store data in all tables. Security components of the architecture must check MT^2 metadata for contracts to see whether the tenant has subscribed functionality related to the base table.

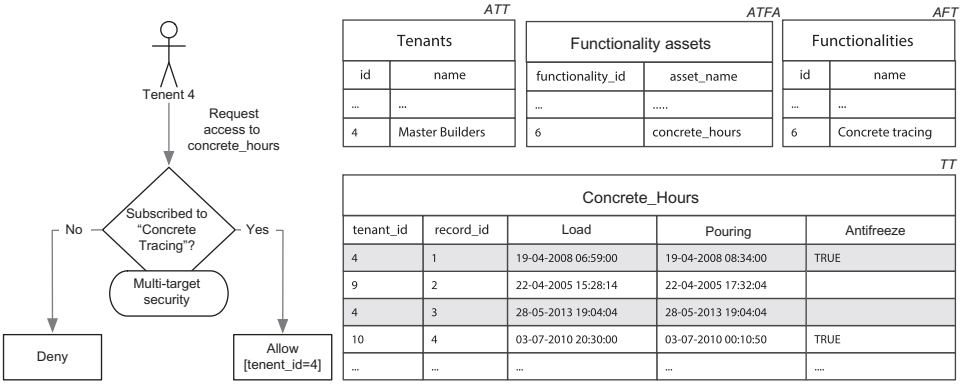


Figure 12. Sample of shared tables structure and security access control.

As example, in Figure 12, table *Concrete_Hours* is a tenant table belonging to the *concrete tracing* functionality (presumably subscribable by constructors). When tenant 4 (see ATT table in the figure) tries to access the table, security components need to check ATFA relation before granting permission. As showed in Figure 11 on the MT^2 metadata, tenant 4 is subscribed to this functionality so the first part of the security check is positive. Once allowed, MT data privacy must ensure that the tenant reads or updates just its records (marked in grey) in addition to storing new ones with tenant_id = 4.

5.3. *Extension model*

Tenants should be able to add personalised fields to their data by means of the extension model of the database. Similarly to the base model, modifications have to be made to security layer in order to allow extensibility just for those functionalities contracted. Figure 13 depicts our approach on the basis of a modified version of the *name–value pairs* technique explained before. Extensibility is reached by using two different tables. In the *extension names table (ENT)*, tenants define customised fields that are stored in the *extension values table (EVT).* In a multi-target environment, the ENT table needs to be

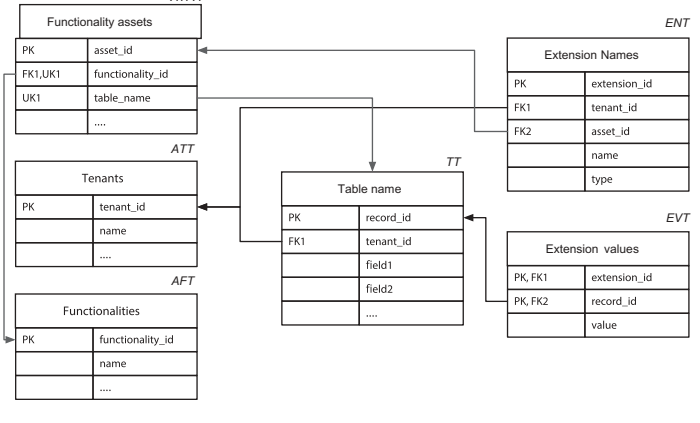


Figure 13. MT^2 extension data model.

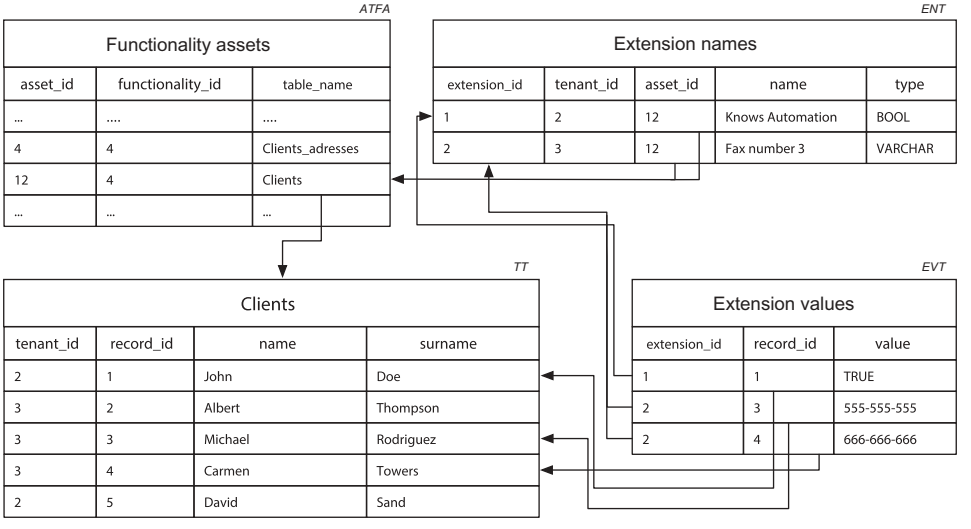


Figure 14. MT^2 extension model sample. Adding customised fields.

Clients T_2

record_id	name	surname	knows automation
1	John	Doe	TRUE
5	David	Sand	

Clients T_3

record_id	name	surname	Fax Number 3
2	Albert	Thompson	555-555-555
3	Michael	Rodríguez	666-666-666
4	Carmen	Towers	

Figure 15. Logical client tables for tenants 2 and 3.

linked to ATFA so that security services control and prevent unauthorised access and secondly to find out which physical table from the functionality is being extended.

Figure 14 depicts an example of this approach. Suppose that tenants 2 and 3 are subscribed to the 'Client Management' feature with *functionality_id* 4. In the ATFA, we can find two base tables for it: *Client_addresses* with *asset_id* 4 and *Clients* with *asset_id* 12. This last table *Clients* does not include some useful fields for tenants such as the *knowledge for automation* (business domain) for tenant 2 and an *extra fax number* for tenant 3; therefore, the data model needs to be extended. To this end, in the ENT table both definitions are stored and then linked to ATFA so that application knows that they should be applied to *Clients* table. *Record_id* field determines to what specific client within the tenancy the values belong. Logical tables are illustrated in Figure 15.

6. Case study

Globalgest (Rico 2013) is an example of a commercial SaaS solution with an underlying MT^{2}A. In Globalgest, modules with a minor level of granularity make up functionalities; this way tenants can select a subset from the portfolio more suitable for its needs. Module examples are c*lient management, billing* or *quoting.* Actual portfolio of Globalgest

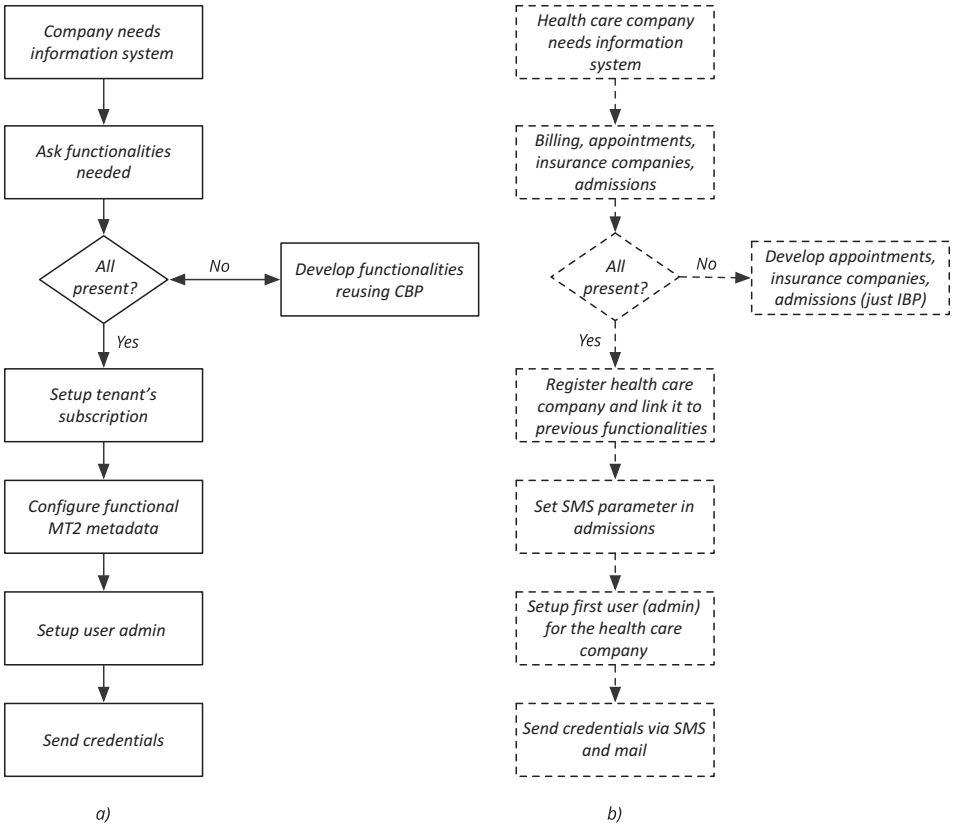


Figure 16. (a) Steps for allocating a new tenant in MT^2. (b) Real case example.

comprises 201 modules that combined together conform the functional portfolio of a particular tenant.

As depicted in Figure 16a, whenever a potential client is interested in the system, Globalgest vendor checks whether functionalities needed by the client can be served with a combination of the current modular portfolio. If not, these features can still be programmed and incorporated into the system. Development of new modules is easier since reutilisation of CBP components reduces the effort to the IBP components. When all functionalities are ready, the vendor sets the tenant's subscription, specifying which modules to deploy and on what terms (configuring the MT^2 metadata). With the functional subscription ready, a first administrator user of the tenancy is created and the application is ready for use.

Right side on Figure 16 instantiates a particular scenario, based on a real case. A health-care clinic wanted a single information system covering the following features: *billing, appointment* management, daily *admissions* control and patients' linkage to *insurance companies*. Globalgest portfolio only covered the first feature, so the rest of them needed to be developed. As there was already a module for *events* management, *appointments* and *admissions* need not to be developed from scratch as they reused former libraries. In addition, control of *insurance companies* resulted to be quite similar to *provider* management, which was previously in the portfolio. Therefore, many CPB components were shared and just the IBP elements needed to be programmed. Once the

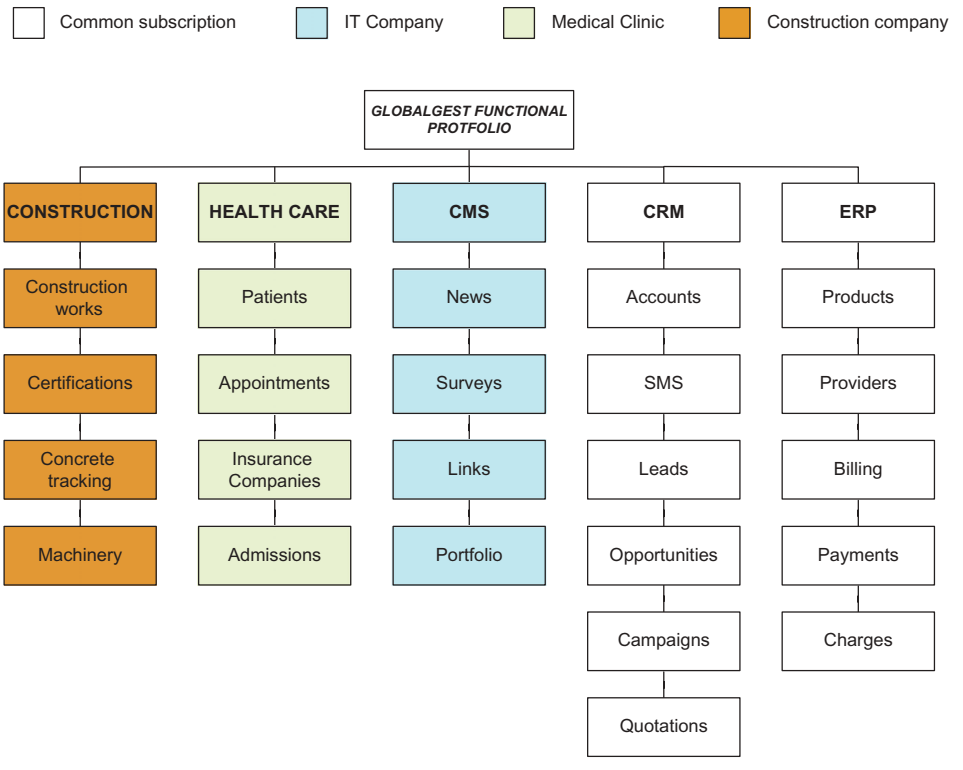


Figure 17. Globalgest serves multiple industries.

development was finished, the clinic company was registered into the system and linked to desired functionalities. Subscription to a specific module may require extra configuration; for instance, appointments module was designed with the possibility of sending SMS reminders to the patients. In this case, the number of purchased text messages was also to be set. With the subscription ready and an administrator user of the company created, passwords were sent via email and SMS.

With the development of new modules, Globalgest not only scales at functional level but also broadens the market of the vendor, as new industries can be covered. With just one instance, Globalgest runs in despair companies such as a medical clinic, a construction company or an IT company.

Figure 17 represents a subset of Globalgest portfolio; white boxes are common subscription for all four companies, the construction company deploys orange boxes (construction related), whereas green boxes belong to the health-care domain and are being served to the medical clinic. IT company needed to control the contents of the company website so CMS functionalities are executed on its tenancy too.

As we see on the previous example, there are common functionalities related to accounting or client management that are likely to be contracted for all industries, whereas others are highly linked to a specific domain.

7. Discussion

Based on the elimination of replications, MT^2 takes a different approach and proposes modifications to MTAs. This modifications work towards obtaining a new kind of SaaS applications with benefits for all parties involved in the industry of software. Since data privacy and security are considered one of the major concerns for companies, in this article we have taken another step and presented an approach for the database model of this novel architecture.

With this extension, vendors reduce costs, decrease time-to-market and have a wider spectrum of potential clients. Besides, expansion to other software sectors is easier and less risky thanks to functional scalability. Customers get lower prices due to a bigger shared scenario, reduce learning efforts due to application unification and have the applications served and portfolio upgraded almost instantly. Programmers on their part find support for agility in development by removing useless replications and have easier tasks of maintenance, as just one system is needed.

MT^2 implements pure multi-tenancy (Bezemer and Zaidman 2010a) at software architecture level; the proposal considers a low degree of isolation environment where tenants from the same instance share machine and database. As in traditional MT, in MT^2 the upper limit of tenants allocated depends on the characteristic of hardware infrastructure, load balancing and the use of performance isolation methods (Li et al. 2008). However, the multi-target profile and the new reduction of prices can make these systems more popular; MT^2 systems are supposed to grow quicker in customers and host more tenants. This situation must be considered; vendors need to be prepared for high scalability.

The feasibility of the proposal is supported by a real development. Globalgest proves how a single application can host several functionalities and deploy them selectively in order to serve companies from different industries (or with different requirements), and yet without multiplying the effort. The experience obtained through Globalgest has revealed shortcomings for the approach:

- **Administrative level** requires higher design and development effort for managing multiple functionalities.
- **Delicate upgrades** in CBP components. Modifications to common component must be painstakingly considered because they affect all functionalities and all tenancies.
- **System performance**:
  - Large number of base tables: A new functionality involves at least one new table in the base model. MT^2 systems with a large portfolio may involve an elevated number of base tables.
  - MT^2 systems are supposed to host more tenants than the MT ones. This can be solved with the elasticity provided by the Cloud, but incurs in costs increases.

8. Conclusions and future work

MTAs are a key technology for the success of SaaS as a new model for software delivery in CC. Multi-tenancy allows multiple customers to be consolidated into the same operational system. This way, software vendors can leverage economies of scale by serving their clients with the same application instance. Clients on their part benefit from a considerable price reduction since they share expenses as well.

Although multi-tenancy can be implemented by other technologies such as virtualisation, in this article we have focused at application level, by means of software architectures that support this shared scenario. When studying MTAs, two tiers must be considered: an administrative tier to manage the rental of the applications' usage and, on the other hand, the instance tier that represents the functionality that tenants execute. This division is also translated at data layer, with an administrative model for vendors, and the base and extension models for tenants.

Security and system customisation is one of the major factors in companies in order to migrate their legacy systems and move to the cloud. For this reason, data components in MTAs are crucial and must be carefully addressed. An overview of the different approaches and schema-mapping techniques has been presented.

In traditional MT, different functionalities involve different applications. This basically involves price increase (multiple subscriptions) to clients and a narrowed spectrum of potential clients to vendors. Despite the fact that components could be quite often shared across different implementations, they are replicated instead, with the consequent waste of time and money. MT^2 is an extension that enhances traditional MTAs so that different functionalities can be served with just one single application instance.

The MT^2 approach was already introduced in other publications. In this paper, we have gone a step further by presenting a database model to support this multi-functional and hence multi-target profile. The proposal is supported by a real development based on MT^2 (called Globalgest).

The novelty of the approach opens a big field for future studies as the new multi-functional scenario comes with new challenges to overcome. Balance management, for instance: in MT^2, two functionalities with high computational requirements could mine system performance and should not coexist under the same instance. Computer-greedy functionalities are to be scaled out (Rico et al. 2013; Chong and Carraro 2006) to new servers or hosted alongside functionalities with lower computer needs.

Coexistence of functionalities with conflictive requirements is also a subject of reflection. Functionalities are software systems themselves (hosted within the MT^2 system) that can be incompatible to each other. When designing a functionality, selecting certain architectural requirements may exclude some other requirements being considered as the driving forces for the development of other functionalities (Niu et al. 2013). For instance, real-time and system flexibility are incompatible by definition. In this regard, recent studies (Niu, Xu, and Bi 2013; Niu et al. 2013) can be used for the analysis and evaluation of candidate EIS architectures. Considering the present MT2 proposal, cohabitation of functionalities with conflictive requirements might be possible with functional isolation and/or selection of optimal architecture candidates.

The rapid growth of the system in terms of tenants must also be considered. MT^2 systems must be able to serve a presumably bigger number of tenants than mono-target systems. Future studies should also ruminate on performance isolation methods (Li et al. 2008) or database management system mechanisms (Wang et al. 2008).

Internet of things and its role for the integration of outsourced services is also a subject of further study, especially in the case of SCM functionalities where IoT is considered as an enabler of real-time quality management and control in the supply chain (Xu 2011b; Tao et al. 2014).

The MT^2 approach probably best meets the IT needs of small and medium enterprises (SMEs), and future publications need to study this suitability. Benefits should also be demonstrated with empirical data.

Acknowledgement

Antonio Rico thanks the Department of Systems Engineering at Delft University of Technology for collaborative and interdisciplinary research environment during his visitation to carry out part of this research.

Funding

This research work has been partially supported by Desarrollo TIC Ltd., and the Spanish Ministry of Economy and Competitiveness with European Regional Development Funds (FEDER) under the research project TEMADAPT [TIN2012-38600].

References

Ambrust, M., A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, et al. 2009. "Above the Clouds: A Berkeley View of Cloud Computing." Rep. UCB/EECS. 28. Berkeley: Dept. Electr. Eng. Comput. Sci. Univ. California.

Aulbach, S., T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. 2008. "Multi-Tenant Databases for Software as a Service." In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data - SIGMOD '08*, edited by J. T.-L. Wang, 1195. New York: ACM Press. doi:10.1145/1376616.1376736.

Aulbach, S., D. Jacobs, A. Kemper, and M. Seibold. 2009. "A Comparison of Flexible Schemas for Software as a Service." In *Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD '09*, edited by U. C. Etintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, 881–888. New York: ACM Press. doi:10.1145/1559845.1559941.

Bass, L., P. Clements, and R. Kazman. 2003. *Software Architecture in Practice*. 2nd ed. Boston, MA: Addison-Wesley Professional.

Bezemer, C., and A. Zaidman. 2010a. "Challenges of Reengineering into Multi-Tenant SaaS Applications." Delft University of Technology, Tech. Rep. TUD-SERG-2010-012. http://repository.tudelft.nl/assets/uuid:d2e87722-f370-4a38-8d8c-adf686a12bf5/TUD-SERG-2010-012.pdf.

Bezemer, C.-P., and A. Zaidman. 2010b. "Multi-tenant SaaS Applications: Maintenance Dream or Nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE) on - IWPSE-EVOL '10*, edited by A. Capiluppi, A. Cleve, and M. Naouel, 88. New York: ACM Press. doi:10.1145/1862372.1862393.

Buyya, R., C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. 2009. "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility." *Future Generation Computer Systems* 25: 599–616. doi:10.1016/j.future.2008.12.001.

Carlin, S., and K. Curran. 2012. "Cloud Computing Technologies." *International Journal Cloud Computation Services Sciences* 1: 59–65.

Chang, F., J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. 2008. "Bigtable: A Distributed Storage System for Structured Data." *ACM Transactions on Computer Systems* 26 (2): Article 4. doi:10.1145/1365815.1365816.

Chong, F., and G. Carraro. 2006. "Architecture Strategies for Catching the Long Tail What is Software as a Service ?" *Most* 479069: 1–22.

Chong, F., G. Carraro, R. Wolter, M. Corporation, and A. Architecture. 2006. "Multi-Tenant Data Architecture Three Approaches to Managing Multi-Tenant Data." *Architecture* 479086: 1–18.

Coffee, P. 2007. *Busting Myths of On-Demand : Why Multi-Tenancy Matters*. Accessed April 12, 2014. https://developer.salesforce.com/page/File:MythbustMultiT.PDF

Foping, F. S., I. M. Dokas, J. Feehan, and S. Imran. 2009. "A New Hybrid Schema-Sharing Technique for Multitenant Applications." Fourth International Conference on Digital Information Management, ICDIM 2009, University of Michigan, Ann Arbor, MI, November 1–4, 1–6.

Fox, A., R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. 2009. "Above the Clouds: A Berkeley View of Cloud Computing." Rep. UCB/EECS. 28. Berkeley: Dept. Electr. Eng. Comput. Sci. Univ. California.

Goedert, J. 2012. "Clearing the Air around Cloud Computing." *Health Data Management* 20: 44, 46, 48.

Jacobs, D., and S. Aulbach. 2007. "Ruminations on Multi-Tenant Databases." Fachtagung fur Datenbanksysteme Business, Technol. und Web, Aachen, March 5–9.

Kaplan, J. 2007. "SaaS: Friend or Foe? Bus." *Communicable Reviews* 37: 48.

Li, Q., Z. Wang, W. Li, Z. Cao, R. Du, and H. Luo. 2013. "Model-Based Services Convergence and Multi-Clouds Integration." *Computers in Industry* 64: 813–832. doi:10.1016/j.compind.2013.05.003.

Li, Q., Z. Wang, W. Li, J. Li, C. Wang, and R. Du. 2013. "Applications Integration in a Hybrid Cloud Computing Environment: Modelling and Platform." *Enterprise Information Systems* 7: 237–271. doi:10.1080/17517575.2012.677479.

Li, Q., J. Zhou, Q.-R. Peng, C.-Q. Li, C. Wang, J. Wu, and B.-E. Shao. 2010. "Business Processes Oriented Heterogeneous Systems Integration Platform for Networked Enterprises." *Computers in Industry* 61: 127–144. doi:10.1016/j.compind.2009.10.009.

Li, S., L. Xu, X. Wang, and J. Wang. 2012. "Integration of Hybrid Wireless Networks in Cloud Services Oriented Enterprise Information Systems." *Enterprise Information Systems* 6: 165–187. doi:10.1080/17517575.2011.654266.

Li, X., T. Liu, Y. Li, and Y. Chen. 2008. "SPIN: Service Performance Isolation Infrastructure in Multi-Tenancy Environment." *Services Computation* 2008: 649–663.

Liao, C.-F., K. Chen, and J.-J. Chen. 2013. "Modularizing Tenant-Specific Schema Customization in SaaS Applications." In *Proceedings of the 8th International Workshop on Advanced Modularization Techniques - AOAsia'13*, 9–12. Fukuoka: ACM. doi:10.1145/2451469.2451473.

Mell, P., and T. Grance. 2011. *The NIST Definition of Cloud Computing*. NIST Special Publication 800–145 (September), 7.

Neves, F. T., F. C. Marta, A. M. Ramalho, R. Correia, and M. de C. Neto. 2011. "The Adoption of Cloud Computing by SMEs : Identifying and Coping with External Factors." 11a Conferência Da Associação Portuguesa de Sistemas de Informação (CAPSI), (Capsi 2011), Lisbon, October 19–21, 11.

Niu, N., L. D. Xu, and Z. Bi. 2013. "Enterprise Information Systems Architecture—Analysis and Evaluation." *IEEE Transactions on Industrial Informatics* 9: 2147–2154. doi:10.1109/TII.2013.2238948.

Niu, N., L. D. Xu, J. C. Cheng, and Z. Niu. 2013. "Analysis of Architecturally Significant Requirements for Enterprise Systems." *IEEE Systems Journal* PP (99): 1. doi:10.1109/JSYST.2013.2249892.

Prabowo, A., M. Janssen, and J. Barjis. 2012. "A Conceptual Model for Assessing the Benefits of Software as a Service from Different Perspectives." In *Business Information Systems - 15th International Conference, BIS 2012*, edited by W. Abramowicz, D. Kriksciuniene, and V. Sakalauskas, 108–119. Berlin: Springer-Verlag. doi:10.1007/978-3-642-30359-3_10.

Qaisar, E. J. 2012. "Introduction to Cloud Computing for Developers: Key Concepts, the Players and Their Offerings." In *2012 IEEE TCF Information Technology Professional Conference*, 1–6. Ewing, NJ: IEEE. doi:10.1109/TCFProIT.2012.6221131.

Rico, A. 2013. "Software De Gestión ERP Y CRM En La Nube – Globalgest ERP." Accessed August 26, 2014. http://globalgesterp.com/

Rico, A., M. Noguera, J. L. Garrido, K. Benghazi, and L. Chung. 2012. "Multi-Tenancy Multi-Target (MT2): A SaaS Architecture for the Cloud." In *Advanced Information Systems Engineering Workshops - CAiSE 2012 International Workshops*, edited by M. Bajec, and J. Eder, 214–227. Gdask: Springer. doi:10.1007/978-3-642-31069-0_19.

Rico, A., M. Noguera, J. L. Garrido, K. Benghazi, and L. Chung. 2013. "Supporting Agile Software Development and Deployment in the Cloud : A Multi-tenancy Multi-target Architecture (MT2A)." In *Agile Software Architecture*, edited by M. A. Babar, A. W. Brown, K. Koskimies, and I. Mistrik, 269–288. Waltham (MA): Morgan Kaufman.

Rico Ortega, A., M. Noguera, and J. L. Garrido. 2013. "Component-Based Design for Multi-tenant Multi-target Support in the Cloud." In *Enterprise and Organizational Modeling and Simulation - 9th International Workshop, EOMAS 2013, CAiSE 2013*, Vol. 153, edited by J. Barjis, A. Gupta, and A. Meshkat, 146–160. Berlin: Springer-Verlag. doi:10.1007/978-3-642-41638-5_10.

Saracca, C. M., D. Chamberlin, and R. Ahuja. 2006. *DB2 9: pureXml–Overview and Fast Start*, 128. IBM Redbooks. http://www.redbooks.ibm.com/redbooks/pdfs/sg247298.pdf

Shroff, G. 2010. "Multi-Tenant Software." In *Enterprise Cloud Computing – Technology, Architecture, Applications*, 104–114. Cambridge University Press. doi:10.1017/CBO9780511778476.013.

Silva, E., and D. Lucrédio. 2012. "Software Engineering for the Cloud: A Research Roadmap." *Software Engineering* (SBES) 2012: 71–80.

Sultan, N. A. 2011. "Reaching for the "Cloud": How SMEs Can Manage." *International Journal of Information Management* 31: 272–278. doi:10.1016/j.ijinfomgt.2010.08.001.

Tao, F., Y. Cheng, L. Xu, L. Zhang, and B. Li. 2014. "CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System." *IEEE Transactions on Industrial Informatics* 10 (2): 1435–1442. doi:10.1109/TII.2014.2306383.

Tao, F., Y. LaiLi, L. Xu, and L. Zhang. 2013. "FC-PACO-RM: A Parallel Method for Service Composition Optimal-Selection in Cloud Manufacturing System." *IEEE Transactions on Industrial Informatics* 9: 2023–2033. doi:10.1109/TII.2012.2232936.

Wang, Z. H., C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An. 2008. "A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing." In *2008 IEEE International Conference on e-Business Engineering*, 94–101. IEEE Computer Society. doi:10.1109/ICEBE.2008.60.

Weissman, C. D., and S. Bobrowski. 2009. "The Design of the force.com Multitenant Internet Application Development Platform." In *Proceedings of the 35th SIGMOD International Conference on Management of Data - SIGMOD '09*, edited by U. C. Etintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, 889. New York: ACM Press. doi:10.1145/1559845.1559942.

Xu, L. D. 2011a. "Enterprise Systems: State-of-the-Art and Future Trends." *IEEE Transactions on Industrial Informatics* 7: 630–640. doi:10.1109/TII.2011.2167156.

Xu, L. D. 2011b. "Information Architecture for Supply Chain Quality Management." *International Journal of Production Research* 49: 183–198. doi:10.1080/00207543.2010.508944.

Xu, X. 2012. "From Cloud Computing to Cloud Manufacturing." *Robotics and Computer-Integrated Manufacturing* 28: 75–86. doi:10.1016/j.rcim.2011.07.002.

Yaish, H., M. Goyal, and G. Feuerlicht. 2011. "An Elastic Multi-tenant Database Schema for Software as a Service." In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 737–743. IEEE. doi:10.1109/DASC.2011.127.

Zhang, Q., L. Cheng, and R. Boutaba. 2010. "Cloud Computing: State-Of-The-Art and Research Challenges." *Journal Internet Services Applications* 1: 7–18.