Multi-Tenancy Multi-Target (MT^2): A SaaS Architecture for the Cloud

Antonio Rico Ortega[1], Manuel Noguera[1], José Luis Garrido[1], Kawtar Benghazi[1], and Lawrence Chung[2]

[1] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Granada, E.T.S.I.I.T., c/ Periodista Daniel Saucedo Aranda s/n, 18071 Granada, Spain
{antoniorico,mnoguera,jgarrido,benghazi}@ugr.es
[2] Department of Computer Science, University of Texas at Dallas, Richardson, Texas 75083, USA
{chung}@utdallas.edu

Abstract. Multi-tenancy (MT) architectures allow multiple customers to be consolidated into the same operational system. Multi-tenancy is key to the success of Software as a Service (SaaS) by means of a new software distribution formula in which customers share application and costs are indirectly assumed by all of them. However, as traditional applications do, each MT application deploys a single functionality, therefore component sharing between applications only occurs in an ad hoc manner and thereby hindering software reuse. In this paper it is introduced Multi-tenancy Multi-target (MT^2), an extension to MT Architectures for the development and deployment of one single software application encompassing several functionalities. To this end, some new components are added to traditional MT Architectures, thus providing new benefits for software developers, vendors and clients, and which are described by means of real examples.

Keywords: multi-tenancy, cloud computing, software as a service, software architecture.

1 Introduction

Cloud Computing has brought high computational resources to everyone [1], so that small and medium-size software vendors have now the opportunity to access high processing capabilities so far reserved to big corporations. Vendors develop new applications offered through Internet as a service, while customers access them through web browsers anywhere and anytime. This new model of software distribution is called Software as a Service (SaaS) [2]; clients subscribe to vendor's application services and pay for using them [2–4]. Customers afford top-software deployments eliminating initial investment and operational expenses.

Multi-tenancy is becoming a key technology for the success of SaaS since clients reduce the cost of software use by sharing expenditures, whereas software vendors maximize sales profits. Multi-tenancy Architectures (MTA) allow multiple customers (aka *tenants*) to be aggregated into the same application. Tenants share not only application, but also capital and operational expenses [5]. Moreover, tenants are also able to customize their applications both in endpoint presentation and data structure according to their particular needs.

In this context, demand has to be supported by an MTA that allows agile accounts creation in the system. Basically, MTA models have two tiers: administrative and instance; the administrative tier [5] provides the functionalities responsible for creating and managing tenants accounts, while the instance tier hosts the applications that tenants execute according to subscription contracts defined at the administrative level.

Traditional multi-tenant applications are shared among tenants with common functional needs. However, each MT application usually deploys one single functionality and therefore component sharing between applications only occurs in an ad hoc manner at lower levels in the architecture and basic shared components need to be replicated for each application.

In this paper we introduce a proposal called Multi-tenancy Multi-target (MT^2), as an extension to multi-tenancy architecture (from now on we will also refer to it as *mono-target architecture*). MT^2 allows multiple functionalities to be offered in the same operational system. This way, applications are distributed among tenants with different functional needs and vendors can host tenants from heterogeneous market sectors. This multifunctional situation seeks for several benefits: companies are able to subscribe to only one SaaS application; vendors have a multi-target market, broadening the spectrum of potential customers; and developers reach agility by avoiding unnecessary replications.

The rest of the paper is organized as follows. Section 2 provides a background on Cloud Computing, SaaS, and Multi-tenancy technologies. The MT^2 proposal is introduced in Section 3. The architectural design decisions supporting the proposal, and also a real development based on it (called Globalgest), are described in detail in Section 4 and 5, respectively. Next, a Section 6 discusses the relevance of the contribution and related work, and finally Section 7 summarizes conclusions and future work.

2 Cloud Computing Technologies

Cloud computing, and related technologies such as SaaS and Multi-tenancy, are producing a big change in comparison with traditional models for software distribution and use. There is still no common agreement about the definition of *Cloud Computing*, and actually, some authors use Cloud Computing as a synonym of *Utility Computing* [6]: "*A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing platforms on demand, which could be accessed in a simple and pervasive way*".

In this paper, we will use Berkeley's definition by which Cloud Computing is defined as the sum of Utility Computing and Software as a Service [7]. Utility Computing [8] refers to the use of computer resources on demand and it enables a distribution formula for software vendors called *Software as a Service* (SaaS). According to [2] *"The basic long-term vision of SaaS is centred around separating software possession and ownership from its use"*. Unlike its predecessor *Software on Premises*, applications are now installed in a Cloud and accessed over Internet; users are not owners of the software, but consumers of web applications.

Figure 1 shows actors of Cloud Computing. The datacenter (Cloud Provider) serves utility computing to Cloud Users who provide applications on-demand (Software as a Service) to tenants (SaaS users).

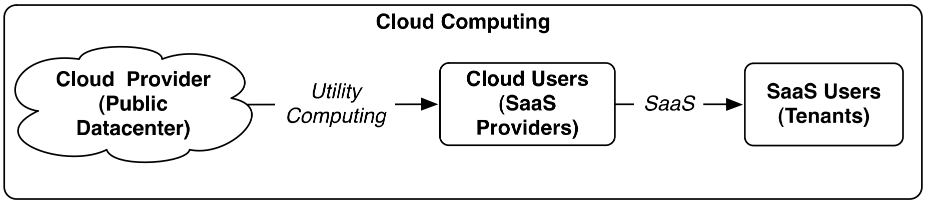

Cloud Computing
Cloud Provider (Public Datacenter)
Utility Computing
Cloud Users (SaaS Providers)
SaaS
SaaS Users (Tenants)

Fig. 1. Actors in Cloud Computing (based on **[1]**)

2.1 Multi-tenancy

SaaS is not the first distribution formula based on outsourcing and software access through wide area networks. However, it is the first in succeeding unlike previous similar attempts like *Application Service Provider* (ASP). Among other reasons, ASP failed because it did not even contemplate the possibility to serve different companies using the same software instance [9] or the ability to provide customized applications [10].

Multi-tenancy is a software architecture that leverages economy of scale by the aggregation of users (tenants) into the same application; software instance is shared among tenants, and so are expenditures.

Figure 2 illustrates multi-tenant system architecture. The lower level tiers perform changes dictated by business layer in both database and file system. Intermediate layers such as presentation or SOA services communicate with browser and smart devices respectively to produce end-users output. Metadata are responsible for system customization so that tenants can get a specific user experience. This customization includes data model extension, adaptation of presentation layer to corporative image and business workflow personalization. Security services must be present in all multi-user systems. In multi-tenant environments, the complexity of this component increases; systems must maintain privacy not only among end-users, but also among different tenants.

Customization and security relay on the model chosen to store data. Several authors have proposed different approaches ([11–16]) of database models in multi-tenancy; though with different terminology, they all agree that the distinction is given by the level of isolation on tenants data [17]. Dean Jacobs, on its article "Ruminations on multi-tenant databases" [5] suggests three approaches:

- Shared Machine: High degree of isolation
- Shared Database: Medium degree of isolation
- Shared Tables: Lower degree of isolation

Regarding the isolation of the database layer, the more isolated the data is among different tenants, the easier to customize, but the more expensive are hardware and maintenance. According to [18], multi-tenancy is pure when using low degrees of isolation (like in the Shared Tables approach); other variations where reutilization of resources is not maximized are considered as semi-multitenant.

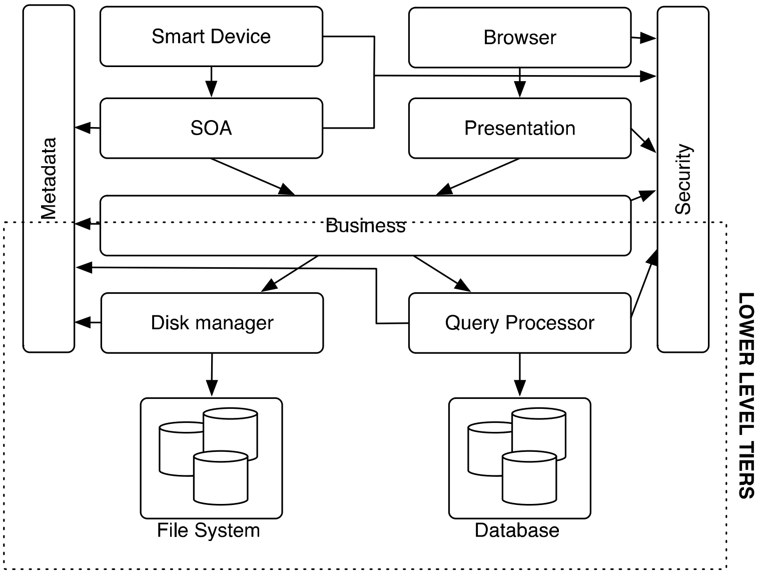


Fig. 2. MT General Architecture model

3 An Approach Extending Multi-tenancy Architecture: MT^2

SaaS applications are highly scalable due to Multi-tenant efficiency [16][17]. Expenses are defrayed among all customers sharing the same software and operational system. As well as instances, users share application functionality. However, current multi-tenancy applications deploy just one single functionality or are aimed to serve a specific line-of-business (*LOB*). With this model, companies have to subscribe to as many applications as services they need. For instance, a company needing *Customer Relationship Management* (CRM), *Content Management System* (CMS) and *Enterprise Resource Planning* (ERP) functionalities would have to contract a different subscription for each of them (see Tenant 2 in Figure 3-a).

In this *mono-target* situation, vendors will have the potential clients spectrum limited by the functionality their applications deploy. For example, CMS vendors will focus on companies needing CMS solutions, whereas CRM vendors will target companies looking for solutions for customer relationship problems.

Traditional multi-tenancy divides the set of potential clients into highly disjoint sets. Companies targeting one set will have to develop new applications if they want to reach other sets of the market. Figure 3-a shows how tenants with different needs subscribe to different applications, since no application can deploy several functionalities (Tenant 2 has subscriptions to three different applications).

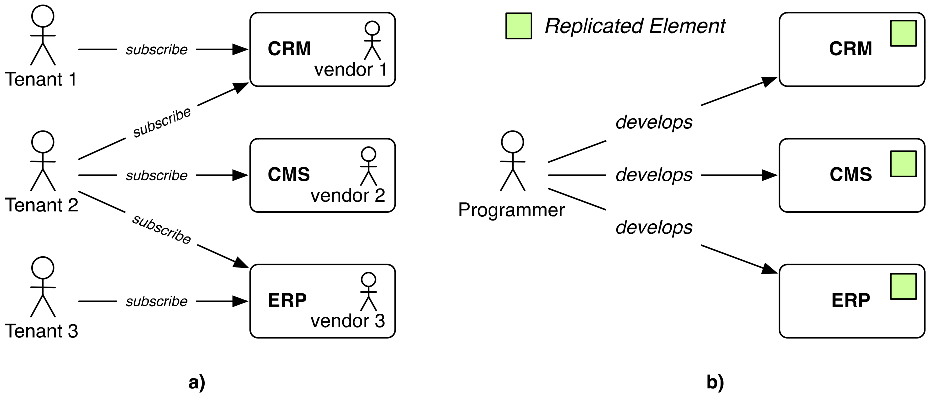


Fig. 3. a) Subscriptions depend on functional needs b) Replication of common development components

At development level, basic features such as user authentication and database connection are common and can be shared in CRM, CMS or ERP applications. Since different software functionalities are hosted in different implementations, these lower level components are to be replicated along all implementations (see Figure 3-b). This replication increases programmers' effort and therefore time-to-market.

3.1 MT^2 Foundations

The main idea behind the MT^2 approach is to allow multi-tenancy systems to deploy not only one single functionality, but several ones. MT^2 extends traditional Multi-tenancy so that tenants with different functional needs could be able to make use of customized end-user applications while sharing the same underlying software system.

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. Tenant subscriptions are defined by a subset of functionalities within the same functional portfolio.

MT^2 systems seek for scalability not only at the tenant level, but also at functional level. Young MT^2 systems may deploy just a few features, but can increase portfolio across the time. Old MT^2 are supposed to have larger functional portfolios, since new functionalities are added on customers' demands and remain on the portfolio, unless outdated.

In this way, companies are able to have multiple functionalities through just one software application. With MT^2, tenants could use just one application to serve all their functional needs. In the last example, the company (Tenant 2) will now be able to unify all its functional needs in just one MT^2 application (see Figure 4-a).

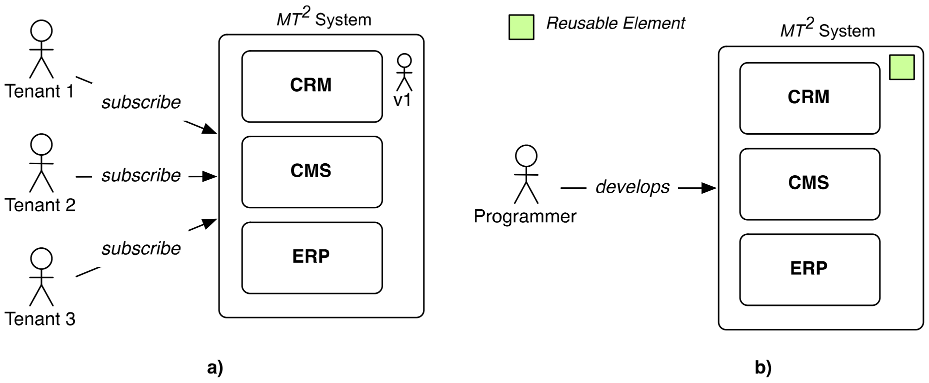


Fig. 4. a) MT^2 systems allow tenants to subscribe to multiple functionalities and broaden the spectrum of potential clients b) Reusability of common resources in MT^2

MT^2 also seeks for to increase the range of potential customers. The perfect separation of potential clients in traditional multi-tenancy disappears and vendors reach a wider range of targets by mixing and overlapping disjoint sets. Figure 4-a shows how a single vendor (v1) can access different markets by offering different functionalities. The larger the portfolio is, the bigger the number of potential customers and opportunities is.

New features involve new sales opportunities. When vendors decide to increase functional portfolio adding *Document Management System* (DMS) functionalities, for instance, chances to raise profits increase. Besides for the demanding tenant, this new development could be available for the rest of tenants (upgrading subscription) and the rest of potential clients in the market.

MT^2 pays special attention to achieve reusability by removing useless replication of common features. In multi-tenancy mono-target, functionalities are deployed in different applications; hence components are replicated. In multi-target, shared components are reutilized among all functionalities reducing time-to-market and development effort. Figure 4-b shows how MT^2 changes replication for resource reutilization.

4 General Model of MT^2 Architecture

Extra components are added to traditional multi-tenant architectures so as to provide a (now) multi-functional subscription. These modifications are present both at administrative and instance level. Figure 5 shows how MT^2A includes those new components (marked in red) on the basis of the MT architecture.

MT^2 is independent of the underlying multi-tenant architecture. Any MT system could be upgraded to MT^2 regardless of design aspects such as the isolation degree in the database. Multi-target extension is based on the reusability of lower level components during development process; many basic features are shared along applications different in nature. In MT^2 all these components (libraries, functions, icons, graphics, style sheets, etc.) are no longer replicated, but reused. Modifications to the traditional MT architecture are mainly focused on:

- Security commitments
- Multi-target metadata for contracts
- Business process reutilization

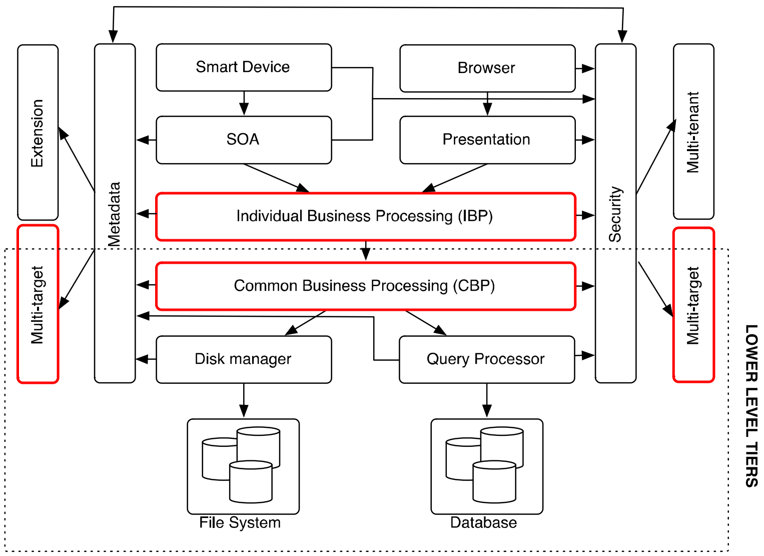


Fig. 5. MT2 Architecture Model

4.1 Security Commitments

Multi-target applications deploy different functionalities depending on tenant's subscription; tenants share application, but functional deployment may differ. In this situation, security components become more complex in architecture, since end-users are allowed to execute those functionalities present in the subscription and not others. Multi-target involves new commitments for security layer at two levels:

- *Tenant level*: Tenants should not deploy functionalities that are not included in subscription. Security must ensure that forbidden functionalities are not deployed.
- *End-user level*: Multi-tenancy applications are multi-user environments at instance level. Tenants end-users have different roles that determine their

capabilities in the system. In MT^2, tenants may have subscription to one functionality, but not all tenants end-users should have access to it. Admin users of the tenant must have the capability to decide for each user what functionalities deploy from the tenant portfolio.

4.2 Multi-target Metadata for Contracts

Multi-target metadata links tenants accounts to functionalities controlling not only functionalities subscribed by tenants, but also contractual features of this relation. For instance, if a tenant wants to subscribe to *SMS* functionality, at least we should set the number of text messages contracted; by setting this parameter in other functionalities such as *Client Management*, does not make sense. Every subscription to functionalities has its own conditions and these are reflected on the Multi-target metadata of each tenant. As well as subscription terms, this component is responsible for:

- Synchronizing with security services and inform about subscription details to prevent access to forbidden functionalities.
- Determining which *individual business processes* to import on *Individual Business Processing* (IBP) layer.

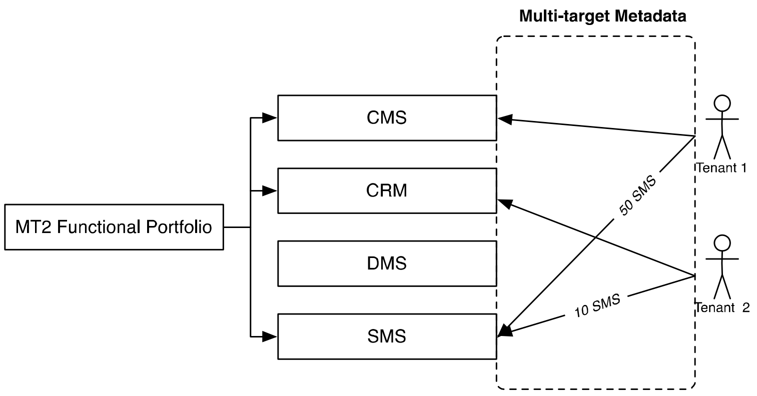


Fig. 6. Multi-target metadata contains subscription details to functionalities

Previous figure shows an example of an MT^2 system with 4 functionalities in the portfolio and two tenants. Subscriptions are defined by Multi-target metadata; both tenants have contracted SMS, but the number of text messages to be sent differs.

4.3 Business Process Reutilization

Reusability of common features along all functionalities is the main cause of this MT^2 extension. In a multi-target environment, business layer is divided into two:

- *Common Business Processing (CBP)*: It contains those elements business-independent and reusable across all functionalities
- *Individual Business Processing (IBP)*: It includes those elements that are business-dependent and which are specifically designed to support one functionality.

During the execution timeline of their application instances, all tenants will import CBP elements statically; however IBP elements will be imported dynamically depending on tenant's subscription. In Figure 7, tenant has a subscription to functionalities F2 and F4. All CBP are imported statically, but just F2 and F4 IBP elements will be imported, since these two functionalities will be deployed in the execution.

CBP represents all those components reused in different functionalities. Features such as privacy or system authentication are no longer to be developed in future functionalities; they are already in the CBP layer. Furthermore, development effort is reduced not only because of reutilization of CBP elements, but also for the extension of them. For instance, if a programmer needs to develop a specific feature and encapsulate it within a class, that class does not need to be developed from the beginning, but it can be coded by extending one existing class from the CBP layer.

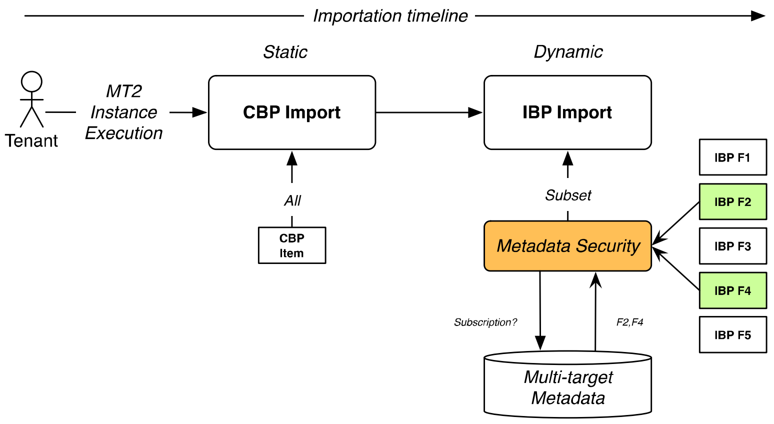


Fig. 7. Static and Dynamic import in MT^2

5 Globalgest, a Software with MT^2 Architecture

Globalgest [19][20] is an example of a business-oriented application based on MT^2 architecture. Installed in 17 companies, Globalgest deploys more than 100 functionalities. Combinations of this portfolio allow Globalgest to serve businesses from different industries such as a medical clinic or an IT company (see Figure 8 and Figure 9). As we see on both figures, tenants share application instance, but do not need to share functionalities or presentation. Globalgest allow tenants to customize application interface by using personalized style sheets and graphics.

Functional portfolio in Globalgest increases on customer demands. Whenever a tenant needs a new functionality, this is developed and included in service portfolio. Existing tenants could subscribe to the new feature and *vendor leads (i.e. potential customers)* could be converted due to this functional improvement. For instance, last functionality developed has been e-commerce connection; this feature (developed ad hoc for one specific tenant) is now available for other tenants who might upgrade their subscription to incorporate on-line selling to their websites.

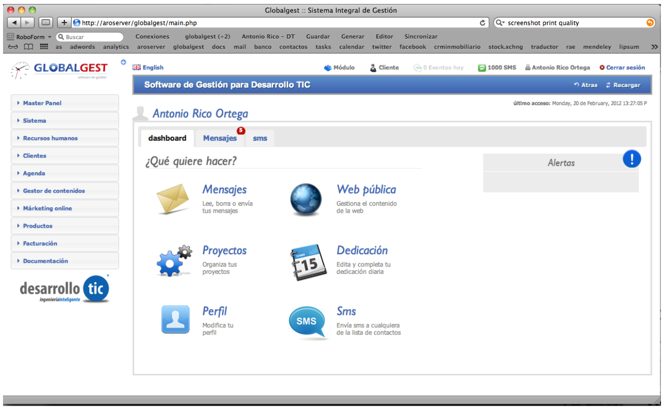


Fig. 8. Globalgest: IT Company Implementation

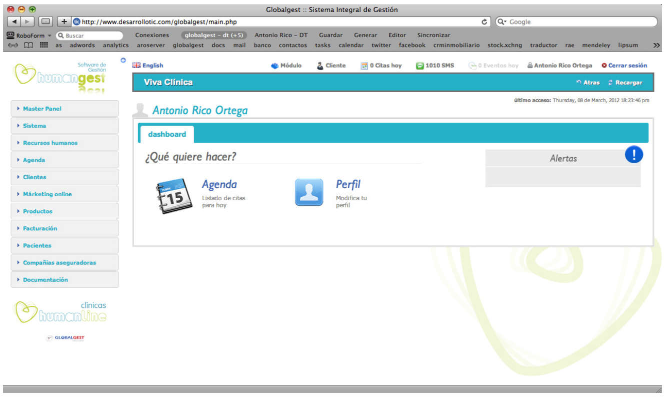


Fig. 9. Globalgest: Medical Clinic implementation

Figure 10 illustrates a diagram with some of the functionalities present in Globalgest portfolio. Green boxes represent the subscription of a medical center, blue boxes are those contracted by an IT company and white boxes represent common subscriptions of them both. In this case, medical's center functionalities were programmed ad hoc for the client, but once developed they remain in the functional portfolio of Globalgest. This means that if another medical company requires them, now these functionalities will be available. This new medical company may not need invoicing, but will likely be interested in patient management, insurance companies monitoring and/or sending programmed SMS reminders for appointments.

Globalgest is real MT^2 software that proves how a single application serves two companies from different industries without duplicating the effort. MT^2 architecture allows Globalgest to deploy and host several functionalities configuring client functional subscription on demand.

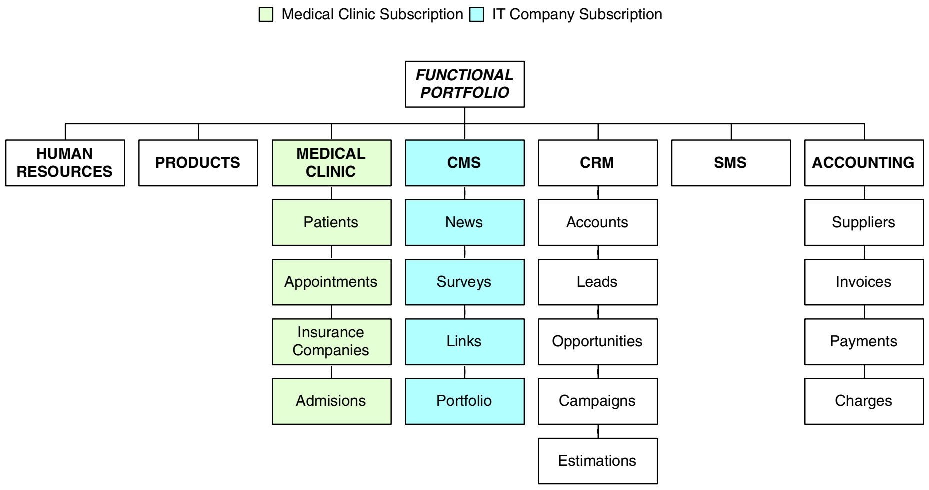
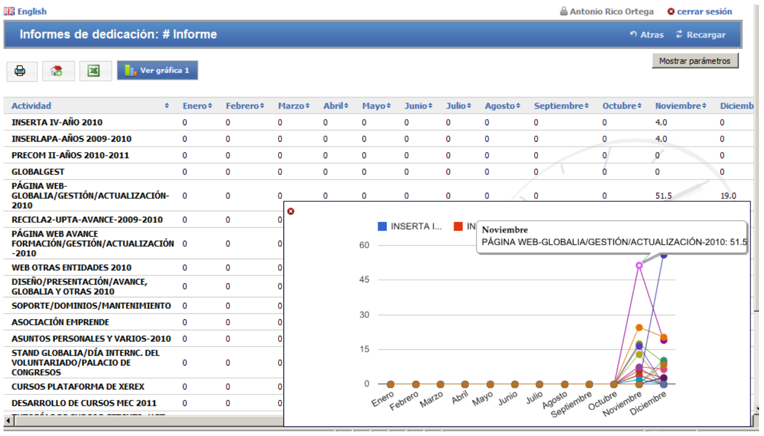


Fig. 10. Medical clinic and IT Company subscriptions


English
Antonio Rico Ortega
cerrar sesión

Informes de dedicación: # Informe

Atras Recargar

Ver gráfica 1

Mostrar parámetros

Actividad	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciemb
INSERTA IV-AÑO 2010	0	0	0	0	0	0	0	0	0	0	4.0	0
INSERLAPA-AÑOS 2009-2010	0	0	0	0	0	0	0	0	0	0	4.0	0
PRECOM II-AÑOS 2010-2011	0	0	0	0	0	0	0	0	0	0	0	0
GLOBALGEST	0	0	0	0	0	0	0	0	0	0	0	0
PÁGINA WEB-GLOBALIA/GESTIÓN/ACTUALIZACIÓN-2010	0	0	0	0	0	0	0	0	0	0	51.5	19.0
RECICLA2-UPTA-AVANCE-2009-2010	0	0	0									
PÁGINA WEB AVANCE FORMACIÓN/GESTIÓN/ACTUALIZACIÓN -2010	0	0	0									
WEB OTRAS ENTIDADES 2010	0	0	0									
DISEÑO/PRESENTACIÓN/AVANCE, GLOBALIA Y OTRAS 2010	0	0	0									
SOPORTE/DOMINIOS/MANTENIMIENTO	0	0	0									
ASOCIACIÓN EMPRENDE	0	0	0									
ASUNTOS PERSONALES Y VARIOS-2010	0	0	0									
STAND GLOBALIA/DÍA INTERNC. DEL VOLUNTARIADO/PALACIO DE CONGRESOS	0	0	0									
CURSOS PLATAFORMA DE XEREX	0	0	0									
DESARROLLO DE CURSOS MEC 2011	0	0	0									

Fig. 11. Worker performance report in Globalgest

As well as features showed in Figure 10, other functionalities implemented in Globalgest and already serving companies are:

- E-Commerce with order management
- Presence Control.
- Workers performance management and reports (Figure 11)
- Financial management and reports
- Teaching companies management: courses, registrations, students, teachers and tutors, classrooms availability, etc.
- Foundation companies management: subsidies, volunteers, human projects.

6 Discussion and Related Work

The market of SaaS Multi-tenancy applications is broad. Clients pay for the use of many different types of software deploying different functionalities such as ERP, CRM, CMS, DMS, etc. Each SaaS application offers a specific functionality to its customers. Clients subscribe to vendor services in a pay-per-use basis. This single-functionality deployment has some inconveniences, not only for customers, but also for software vendors and developers.

When a company needs a particular functionality such as that of a CRM, it compares among all vendors and chooses the one that best fits both its budget and needs. If this same company needs another functionality, e.g, that of a CMS, it will have to repeat the same process by subscribing to another different software application from the same or a different provider. In this case, companies contract as many software subscriptions as functionalities they need. So long as prices increase, users will have to learn the use of several different software interfaces. In this 'mono-functional' context, the spectrum of vendors' potential clients is limited by the purpose each application serves. For example, a CMS vendor will focus on companies needing CMS solutions and not needing other functionalities.

From a developer point of view, there are many basic components that could be shared in applications of different nature. In lower level development, features such as user authentication and database connection are common, regardless of the kind of application (CRM, CMS or DMS). However, these lower level components are to be replicated along all implementations, since different software functionalities are hosted in different application instances.

Software Product Line Approach (SPLA) is also based on the reusability of elements called *artifacts* for delivering affordable customized applications. However, while reusability takes place during development process in SPLA, MT^2 reuses CBP components during execution. Furthermore, we consider in a higher abstraction level of meta applications than that of SPLA mass production. In MT^2, customers not only get to customize their instance, but they can also have a completely different application instantly by changing their subscriptions. Both approaches pursue similar goals and benefits such as reductions of expenses and time-to-market, but exploiting different means.

Multi-tenancy is a novel paradigm. Proposals and implementations are scarce and studies for this area are mainly schema-mapping techniques and benchmarking ([11–17]). Based on the elimination of useless replications, this paper takes a different approach and proposes modifications to the general AMT; MT^2 is an extension applied to the whole architecture rather than just the database layer. This modifications work towards obtaining a new kind of SaaS applications with benefits for all parties involved in the industry of software.

7 Conclusions

Multi-tenancy permits several customers to share network applications in the *Cloud*. This is essential for the success of SaaS as software distribution formula; users are no longer owners of software but tenants of it. In this model, operational and maintenance costs are also allocated along users.

Traditionally, applications aim to serve one single purpose. For instance, CRMs are used for managing customers' accounts and *leads* conversion, but not for other functionalities. Therefore, different purposes involve different applications. There are many basic components that could be reused along all these different implementations, like authentication, payrolls, message broadcasting and notification, etc.; however, developers do replicate it along all implementations. Commercially speaking, traditional Multi-tenancy can be called Mono-target: vendors have their potential clients limited by software purposes. The users themselves have to subscribe to as many services as they need; hence incrementing costs and increasing learning effort.

This paper has presented the MT^2 architecture that is supported by an extension to the current Multi-tenant architectures. MT^2 tries to go a step further in relation to the current MT applications. MT^2 turns traditional Multi-tenancy software into Multi-target applications. MT^2 is based on the elimination of unnecessary replication and the reutilization of lower level components of the architecture. MT^2 applications deploy several functionalities and tenants choose which one to subscribe to. Users unify all applications in one and thus, reduce expenses and effort; vendors broaden the range of potential clients and developers speed up implementation and development.

In order to illustrate the applicability of the proposal, we have introduced Globalgest, a commercial software based on the MT^2 architecture that shows how one individual application can host tenants from different lines of business.

Acknowledgements. This work has been partially funded by CEI-BioTIC Granada under project 20F2/36 and the Andalusian Regional Government under projects SEJ-6765 and TIC-6600.

References

[1] Armbrust, M., Fox, A., Griffith, R., Joseph, A.: A view of cloud computing. Communication of the ACM (2010)

[2] Turner, M., Budgen, D., Brereton, P.: Turning software into a service. Computer 36(10), 38–44 (2003)

[3] Motahari-nezhad, H.R., Stephenson, B., Singhal, S.: Outsourcing Business to Cloud Computing Services: Opportunities and Challenges. Development (2009)

[4] Dubey, A., Wagle, D.: Delivering software as a service. McKinsey Quarterly (May 2007)
[5] Jacobs, D., Aulbach, S.: Ruminations on multi-tenant databases. In: Fachtagung für Datenbanksysteme in Business, Technologie und Web, Aachen, Germany, pp. 5–9 (March 2007)
[6] Wang, L., Tao, J., Kunze, M., Castellanos, A.C., Kramer, D., Karl, W.: Scientific cloud computing: Early definition and experience. In: 10th IEEE International Conference on High Performance Computing and Communications, HPCC 2008, pp. 825–830 (2008)
[7] Armbrust, M., et al.: Above the clouds: A berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28 (2009)
[8] Parkhill, D.F.: Challenge of the Computer Utility, p. 232. Addison-Wesley Educational Publishers Inc. (1966)
[9] Liu, G., Jiang, H., Geng, R.: Software design on a SaaS platform. In: 2010 2nd International Conference on Computer Engineering and Technology, pp. V4-355–V4-358 (2010)
[10] Papazoglou, M.P.: Service -Oriented Computing: Concepts, Characteristics and Directions. Information Systems
[11] Aulbach, S., Jacobs, D., Kemper, A.: Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques. Techniques, 1195–1206 (2008)
[12] Aulbach, S., Seibold, M., Jacobs, D., Kemper, A.: A Comparison of Flexible Schemas for Software as a Service. Acme, 881–888 (2009)
[13] Chang, F., Dean, J., Ghemawat, S.: Bigtable: A distributed storage system for structured data. ACM Transactions on (2008)
[14] Copeland, G.P., Khoshafian, S.N.: A decomposition storage model. ACM SIGMOD Record 14(4), 268–279 (1985)
[15] HBase Storage Architecture (2009), http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html
[16] Chong, F., Carraro, G.: Architecture Strategies for Catching the Long Tail What is Software as a Service? Most 479069, 1–22 (2006)
[17] Chong, F., Carraro, G., Wolter, R., Corporation, M., Architecture, A.: Multi-Tenant Data Architecture Three Approaches to Managing Multi-Tenant Data. Architecture 479086, 1–18 (2006)
[18] Bezemer, C.-P., Zaidman, A.: Challenges of Reengineering into Multi-Tenant SaaS Applications. Challenges (2010)
[19] Rico, A.: Globalgest SaaS - Software as a Service, http://globalgest.info/ (accessed November 01, 2011)
[20] Rico, A.: Desarrollo TIC. SEO, Web, and Software Development, http://www.desarrollotic.com/ (accessed February 15, 2012)