Doctoral Thesis

PhD Program on Information and Communication Technologies

Multi-tenancy Multi-target (MT^2): An Extension to Cloud Multi-tenant Architectures for Multi-service support in SaaS Enterprise Information Systems

---

ugr | Universidad de Granada

Departamento de Lenguajes y Sistemas Informáticos

Author

Antonio Rico Ortega

Supervisors

José Luis Garrido Bullejos

Manuel Noguera García

2015

Editor: Universidad de Granada. Tesis Doctorales
Autor: Antonio Rico Ortega
ISBN: 978-849125-880-3
URI: http://hdl.handle.net/10481/43793

El doctorando Antonio Rico Ortega y los directores de la tesis José Luis Garrido Bullejos y Manuel Noguera García garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

En Granada, a 19 de Octubre de 2015.

Directores de la Tesis

José Luis Garrido Bullejos Manuel Noguera García

Doctorando

Antonio Rico Ortega

The PhD candidate Antonio Rico Ortega and the thesis supervisors José Luis Garrido Bullejos and Manuel Noguera García guarantee, by signing this thesis, that the work has been done by the PhD candidate under the guidance of the directors of the thesis and, as far as our knowledge reaches, in the realization of the work, the copyrights of the cited authors have been respected, when their results or publications have been used.

Granada, October 19, 2015.

Thesis Supervisors

José Luis Garrido Bullejos Manuel Noguera García

PhD Candidate

Antonio Rico Ortega

A mis padres Juan and Araceli

A mi mujer Carmen y mi pequeño Carlos

Os quiero mucho

Agradecimientos

En primer lugar quiero expresar mi más sincero agradecimiento a los doctores José Luis Garrido y Manuel Noguera por la dirección de esta tesis. Parece un tópico que diga que sin ellos no hubiera sido posible, pero es la pura verdad; me habéis empujado desde el principio. Siempre tendré grabado en mi retina el día en el despacho de José Luis en el que me animasteis a investigar y profundizar en mi idea para así completar el trabajo de fin de máster. Desde entonces ha pasado de todo, y no sólo ha sido trabajo; además de horas y horas de estudio, artículos, congresos y reuniones… también hemos disfrutado de relajantes cervezas repletas de risas así como de estupendos encuentros familiares que siempre estamos intentando repetir. Hemos visto incluso crecer nuestras familias con el nacimiento de nuevos miembros que nos han bendecido. Esta tesis más que un camino de esfuerzo, ha sido una experiencia alucinante: **gracias**.

I want also to thank Doctor Joseph Barjis for his incalculable and disinterested help provided during my stance in Delft. My dear Joe, I came back from Holland not only with an enriching academic experience, but also with a new friend who opened his family door to my own. I sincerely hope this journey together does not end here; remember that you have a family in Spain too. No matter how long this paragraph is, it will never make up for your altruism, kindness and all the things that you did for me.

Me gustaría también agradecer el apoyo prestado por todos mis compañeros del Departamento de Lenguajes y de Sistemas informáticos de la Universidad de Granada y en especial a los miembros del grupo MYDASS Kawtar, María José, Miguel, Mavi y Marisa. Dentro de este conjunto enfatizo sobre os que considero amigos, colegas y acompañantes de fatigas en proyectos comunes: Carlos, Tomás y Álvaro, muchas gracias.

No quiero terminar sin citar y expresar mi gratitud al Dr. Lawrence Chung y al Dr. Mamadou Seck, por sus valiosas aportaciones.

Por último, mi más sinceras disculpas a mi familia por el tiempo que esta tesis nos ha "robado". En especial a mi mujer Carmen y a mi pequeño e "inquieto" Carlos. Os quiero mucho.

Resumen

Multi-tenancy (MT) es un patrón de arquitecturas software que posibilita que varios usuarios compartan la misma instancia de una aplicación y por tanto, los gastos asociados con la implantación y el mantenimiento de la misma. Esta reducción de costes, hace que las Arquitecturas Multi-tenant (AMTs) sean consideradas como uno de los pilares fundamentales en el éxito de *cloud computing* (computación en la nube) y de su modelo de distribución de software *Sofware-as-a-Service* (SaaS). En SaaS, las aplicaciones pasan de ser compradas e instaladas localmente para ser "consumidas" como un servicio accediendo a ellas a través de la web. Del mismo modo, los usuarios dejan de ser propietarios para convertirse en arrendatarios (*tenants* en inglés) de las aplicaciones.

Este nuevo paradigma es especialmente interesante para las pequeñas y medianas empresas (*pymes*), no solamente por los claros beneficios económicos, sino que también por el rápido acceso a soluciones software de calidad, hasta ahora reservadas para las grandes compañías. Por otro lado, SaaS favorece también a las empresas desarrolladoras que aprovechan las capacidades cloud tanto a nivel técnico como comercial para aumentar su espectro de clientes potenciales y/o disminuir el *time-to-market*.

A pesar de todas las ventajas de SaaS, son pocas las empresas que han decidido adoptar este modelo. Aún con la rebaja de precios, todavía existen empresas que no pueden permitirse este tipo de aplicaciones o la consideran inadecuada. Además, el entorno compartido que ofrecen las AMTs provoca recelos para su contratación ya que la *seguridad* y la *privacidad* son los factores que las organizaciones ponderan con mayor preocupación. Es por ello por lo que los arquitectos software deben prestar una especial atención a la capa de datos, integrando mecanismos de seguridad eficaces que aseguren la privacidad y eviten accesos indeseados.

Al igual que ocurre en las soluciones software tradicionales, las aplicaciones SaaS empresariales despliegan una única funcionalidad. Así, necesidades funcionales no presentes en una aplicación implican la contratación de otra/s solución/es software o de pasarelas de integración entre los distintos sistemas (probablemente de un proveedor/es distinto). Esta situación no sólo supone un perjuicio económico para el comprador (por

las nuevas suscripciones y necesidades de integración) sino que también incrementa la complejidad general y termina minando el uso de las aplicaciones. Por su parte, las empresas de software deben acometer y desarrollar nuevos proyectos si quieren cubrir y dar servicio a estas demandas funcionales del mercado; si con su portfolio no son capaces de servir completamente a las empresas, tendrán que desarrollar nuevas aplicaciones. Dentro de estos nuevos desarrollos, a pesar de que muchos componentes arquitectónicos son idénticos y podrían ser *reutilizados*, éstos son sin embargo *duplicados* y reconfigurados en cada aplicación resultante. El actual escenario con múltiples soluciones mono-funcionales afecta negativamente a un soporte *ágil* en el desarrollo de software, incrementando el esfuerzo y *el time-to-market*.

El entorno compartido para múltiples usuarios que proporciona SaaS convierte a su AMT subyacente en un elemento de especial importancia. Sin embargo, existen pocos estudios que profundicen en las AMTs detallando el diseño en base a componentes y relaciones. Los trabajos actuales, además de escasos se centran en el nivel de aplicación y no tienen en cuenta mecanismos de integración ni un nivel de administración necesario para vendedores. A nivel de datos, a pesar que existen varios trabajos que explican diversos enfoques, nuevamente los autores se centran en el nivel de aplicación ejecutado por los usuarios y no en la gestión y administración de estos sistemas.

Esta tesis estudia en profundidad las arquitecturas cloud multi-tenant definiendo modelos de referencia detallados para las AMTs y su capa de datos, lo cual no se encuentra descrito de forma explícita en la literatura. Además, partiendo de dichos modelos se propone una extensión que permite a las AMTs tradicionales desplegar múltiples funcionalidades de forma selectiva y en una única instancia de software. Esta extensión busca fomentar la reusabilidad entre componentes, reducir nuevamente costes y disminuir la complejidad en el uso de sistemas. Además, nuestro enfoque aboga por las *arquitecturas ágiles* ya que proporciona un soporte a la agilidad en el desarrollo que reduce esfuerzos y acorta el time-to-market. Por último, como demostración de la aplicabilidad de la propuesta se presenta *Globalgest*, una aplicación comercial con arquitectura AMT extendida que actualmente da servicio a más de cuarenta empresas pertenecientes a diferentes sectores y/o con necesidades funcionalidades distintas.

Abstract

Multi-tenancy is an architectural pattern that permits several customers to share and run the same application instance, hence reducing the overhead via amortization over several organizations. This costs reduction has made the Multi-tenant Architectures (MTAs) to be considered one of the cornerstones in the success of *cloud computing* and its software distribution formula *Software-as-a-Service* (SaaS). In SaaS, applications are no longer purchased and installed locally, but "consumed" as utility and accessed via Internet. Similarly, users cease to be owners to become *tenants* of the applications and pay regularly for its use.

This new paradigm is especially attractive to *Small and Medium Enterprises* (SMEs), not only because the clear economical reasons, but also because SaaS enables the access to top quality applications, so far reserved to big corporations. Furthermore, SaaS benefits also development companies that leverage cloud capacities not only at technical level, but also at commercial to broaden the spectrum of potential clients and/or reduce the *time-to-market*.

However, the path to SaaS adoption by SMEs is subtle and subject for scientific research. Despite the price reduction, many small companies find this formula still unaffordable or inadequate. Besides, organizations are reluctant to shared environments since *privacy* and *security* are major concerns; confidence in the system relies on them and any breach could undermine the use of the system. For this reason, the data layer of traditional MTAs has attracted serious attention in the literature. Software architects must pay particular attention to security and integrate efficient mechanisms and components so as to ensure privacy and unauthorized entries.

As traditional on-premises software does, SaaS MT applications deploy a single functionality. Therefore, functional needs that are not present in the system entail new subscriptions to other application/s or integration services between systems (probably from a different vendor/s). This scenario affects not only economically to companies (due to the new fees and the integration costs), but also increases users' complexity and learning effort. On the other hand, development companies must carry out new projects and program new applications in order to satisfy the functional demands of the market.

These new solutions *duplicate* common architectural components across different implementations, components that could be otherwise *reused* on the basis of appropriate design decisions. Consequently, the current scenario with multiple mono-functional solutions hinder the support to *agility*, as these duplications increment the effort and time-to-market.

SaaS shared environment makes its underlying AMT an artifact of critical importance. However, profound studies detailing components and relations in this architecture are scarce. Current literature, besides infrequent, focuses at application level and does not pay attention to the indispensable administrative level used by software vendors or to the integration mechanisms with other systems. At the data layer design, publications are more abundant but once again, approaches focus on customers' level and not in the management and support required to administrate this kind of applications.

This thesis studies cloud multi-tenant architectures in depth, and explicitly define detailed reference models both for the AMTs and its critical data level. On the basis of these models we propose a novel extension to the traditional AMTs that allow multiple functionalities to be deployed selectively using one single software instance. This extension aims to foster reusability, again reduce the prices and decrease complexity in software use. Moreover, the approach advocates for *agile architectures*, since it provides a support for the *agility in development*, minimizing the effort and shortening the time-to-market. Lastly, in order to demonstrate the applicability of the proposal we present *Globalgest*, a commercial software application with underlying extended MTA architecture. Globalgest can deploy selectively multiple functionalities and it is currently serving more than 40 companies belonging to different sectors.

Índice general

Índice de figuras

Índice de tablas

Preface

1 Introduction

There is a common perception to consider the future of computing as the 5th utility [1]–[4]. Together with water, gas, electricity and telephony we should be able to access computing on demand and pay regularly for this consumption. In cloud computing, cloud providers serve computation as a commodity whilst cloud clients are able to access these resources on demand. This new paradigm brings a kind of elasticity of resources that is unprecedented in the history of IT [4]; heavy batch-oriented tasks can be solved as quickly as programs can scale: for instance, using 1000 servers during one hour cost no more than using one server for 1000 hours. Thus, cloud computing has become one of the top priorities of many CIOs (Chief Information Officers) and is rapidly moving from early adopters to mainstream organizations [5].

Software as a Service (SaaS) is considered one of the three main service models in cloud computing [6]; the other two are Infrastructure as a Service (IaaS), which involves access to primary resources, and Platform as a Service (PaaS), which allows software developers to deploy their own applications without being aware of infrastructure issues. 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 [7], [8]. In this SaaS model, companies do not need to invest in either IT infrastructure, acquisition or development of Enterprise Information Systems (EIS); besides, operational costs are distributed over all tenants and hence **expenses are notably reduced** [9].

A software architecture describes a set of system components as well as their topological relations [10]. A significant contribution to software architecture is the definition of patterns that can be used as guides and blueprints when designing a software application [11]. In the very core of SaaS success lays an innovative architecture called *Multi-tenant Architecture* (MTA).

Multi-tenancy (MT) is an architectural pattern that permits several 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.

MT refers to the ability that one system has in order to serve multiple tenants with just one software instance. Although can be implemented at system level by means of other technologies such virtualization, in this research we discuss the application level where multi-tenancy is implemented **trough software architecture**

The multi-tenant model is considered an essential characteristic for cloud computing and its software delivery model [6], [12]. In [13], three attributes are usually identified to be considered in a good SaaS application: scalability, configurability and multi-tenant efficiency.

Thus, MTAs provide SaaS applications with the ability to serve many customers through a single instance of software. Moreover, tenants are also able to **customize** the application to their particular needs. The use of metadata let tenants to tailor the system in three different levels: database model, user interface and business logic [14]. This way, multi-tenancy seems to be transparent for customers giving the impression that they are running a dedicated solution.

MT architecture leaves one major concern regarding the companies' data, their **privacy and security**. In shared environments like SaaS, these aspects 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. Though with different denomination, literature agrees to categorize multi-tenant databases depending on the isolation degree on tenants' data. While *isolated* approaches store tenants data separately, a *shared* approach will host tenants data into the same location.

In multi-tenancy, **component design** for system scalability and performance, data privacy and application customization must be carefully meditated. In contrast to traditional multi-instance applications, MTAs need to provide an administrative level on top of the architecture for managing the multi-tenant character of the application [15], [16]. Basically, MTA models need two tiers: administrative and instance; the administrative tier 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 (Figure 1).

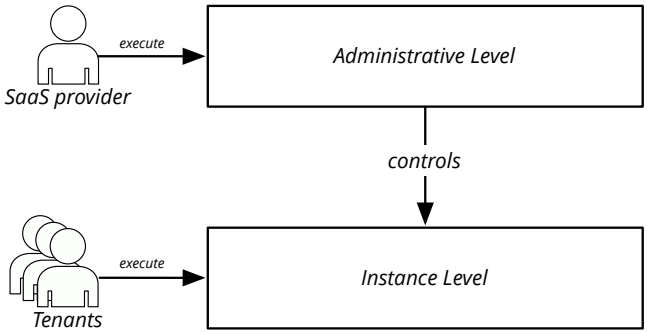


Figure 1. Levels in Multi-tenant Architectures

Agility has been widely advocated in the past few years as a development philosophy that improves efficiency in software construction [17]. Most agility methods and techniques focus on the organization of the members in software teams and the extensive adoption of demonstrated software engineering best practices, such as code refactoring. *Agilists* consider software architecture something "evil" from the past, a bad habit that only carries tons of documentation, big up-front design (BUFD), and *you ain't gonna need it* (YAGNI) [18]. However, as well as these claims, agile methods strive to deliver working and valuable software early and often to clients. If so, it would be worthy to start from a proved and supportive architecture rather than starting from scratch.

Recently, ruminations on this support have provoked a contemporary debate about the coexistence of agility and architecture. Prominent authors from the System and Software Engineering communities have claimed and advocated in favor of *agile architectures* [19]–[21]**.** For instance, in *software on premises* business model, the demand for applications is met through mass production of computer-readable formats that are sold by traditional software retailers. In contrast, SaaS applications are in the cloud and demand has to be supported by architectural styles that allow rapid subscription and configuration. In a cloud multi-tenant environment, where easy scalability is key, architectures **supporting agility in deployment** and rapid provisioning become critical; there should be a fast change to the system so as to accommodate potential tenants in as short period of time as possible. In this regard, architecture should not be doomed to a forbidden non-agile malpractice, but as **helpful assets** and supportive tools.

Multi-tenancy improves scalability since new tenants can be easily allocated into the system; this reduces costs and opens software providers **new market opportunities**. This new share is known as the *Long Tail* and it is based on the sense of *selling less of more* [22]. Chris Anderson conceived the original idea but its applicability to software architecture was studied after the disruption of multi-tenancy. In [23] the authors explain how for every big corporation purchasing a software system, there would be hundreds of small companies willing to use the same application. Lowering the price of applications and selling to the "tail" of the market instead of the head is the reason of the success of SaaS.

With a mature SaaS multi-tenant model, **everybody wins**: clients reduce the cost of software use by sharing expenditures, software vendors maximize sales profits by reaching larger markets whereas developers find support in deployment [14].

To sum up, cloud computing has enabled a new strategy for software distribution. Applications are no longer purchased, but consumed as a utility on a pay-per-use model called SaaS. Multi-tenancy is the architectural pattern that enables the benefits on which success of SaaS is grounded. MT applications support the agility in deployment and are highly scalable; this way, vendors are able to allocate many customers with one solution and reach a more populated market. Customers for their part see general costs reduced, as expenditures are pooled among tenants (Figure 2).

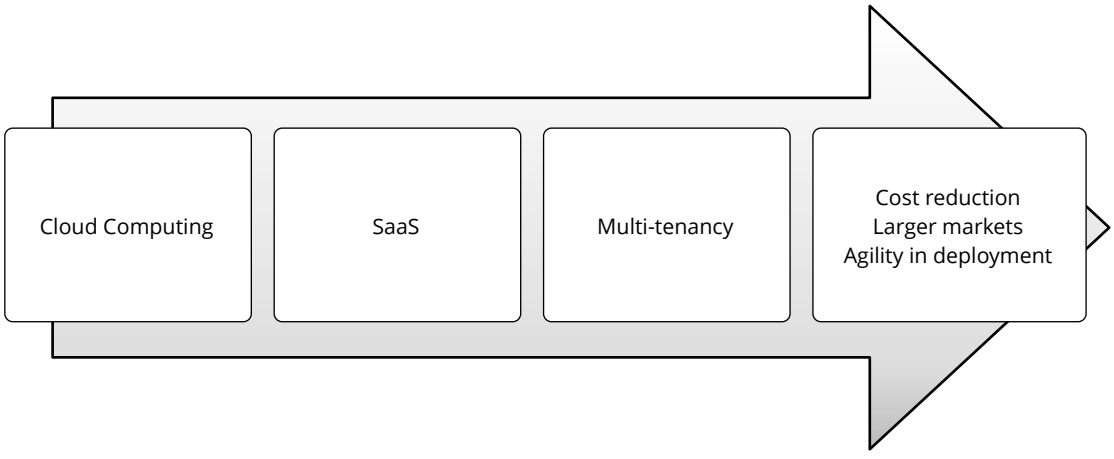


Figure 2. Steps to benefits in SaaS MT

2 Motivation

The market of 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 spend more money**, as they need to subscribe to as many applications as their business activities need to be supported. For example, a company needing a CRM will compare in the market among those providers serving CRM functionalities and subscribe to one of them. If this same company would need a CMS or ERP, it will probably end up by subscribing to another service from a different SaaS vendor.

Therefore, software providers **target just one SaaS niche** per application, depending on the functionality deployed. Broadening the spectrum of potential clients entails the development of new MT applications serving different functionalities. For example, CRM vendors can only target those companies needing to control and improve the relationships with their respective clients. Pointing at companies demanding control of contents of their corporate websites should involve the design and development of new CMS solutions.

With SaaS, costly applications should now be accessible to Small and Medium Enterprises (SMEs) since cost diminution is considered one of their major reasons for cloud adoption [24]–[26]. However, the pathway to SaaS adoption by SMEs is subtle and subject for scientific research [26]–[29]; despite this price decrease, SaaS embracement by small companies is not as widespread as expected. Size of SMEs dramatically varies; companies with 200 employees fit in the range but companies with 10 employees do fit as well. Small companies may have not the capital resources or the IT knowledge so as to benefit from this new distribution formula. Furthermore, even if

one SME can afford the solution, it is normally designed for bigger companies and the deployed functionality far exceeds its needs.

That's for customer companies, but what about vendors? If we try to find SaaS providers belonging to the lower extreme of SMEs, we will rapidly find out that they are scarce or practically inexistent. In an extended sense of SMEs, larger firms have greater propensity to benefit from cloud information systems [27], but many small companies find them **unreachable**.

From the **developer's** point of view, current multi-tenancy **increases delivery times and effort**. Many MT applications could share common lower development components of their architecture and nevertheless they maintain exact copies. Database connectors, user authentication or general user interface components (graphics, style sheets, etc.) among others, might be the same in multiple SaaS applications, whether it is for a CRM, CMS or ERP. However, all these common components are eventually **duplicated** over all these different implementations.

For instance, let us consider the connection process to an ERP or a CMS. In both applications, end users input the username and password, and then these credentials are used internally so as to grant or deny access. In this case, the programming would be exactly the same regardless the functionality deployed or the industry for which the software is designed. Nonetheless, developers have to clone and connect identical components in both software implantations. Moreover, these redundancies multiply effort times when it comes to maintenance tasks.

In terms of agility, duplication of common architectural components for each multi-tenant application means **losing a valuable time**. During development process, developers will have to copy and adapt/reconnect components over implementations; vendors serving different applications will have to manage different MT applications with even duplicated records of the same multi-subscribed customers; moreover, customers on their turn will have bigger learning effort as different applications need to be learned and used.

Cloud computing is a long held dream to provide high computational capabilities to everyone. Indeed, its traditional SaaS multi-tenant model delivers important benefits,

but **still has some drawbacks**. Particularly, in the current multi-tenant situation customers see the costs increased or still unaffordable; vendors reduce the addressable market and programmers waste efforts on duplications. These inconveniences mine the general use of cloud applications as they affect directly to all the actors involved in cloud computing.

The **hypothesis** on which we build our research is rather simple, but it aims to be powerful. Duplication of common components across applications is the main cause of the inconveniences detected in current multi-tenancy. Instead, if we share and reuse those elements by unifying multiple applications in one, previous disadvantages would be overcome. Sharing has been the key in the success of cloud computing and SaaS and sharing will be the solution to multi-tenancy weaknesses.

Therefore, the study and research of MTAs is fundamental, as these architectures are elements of critical importance for the sharing environment that has been the key in cloud computing and SaaS. However, there have been little works detailing components and topological relations for MTAs. If any, research from current authors is **focused on the application level** and it does not pay attention to the indispensable administrative level used by software vendors. Furthermore, to the best of our knowledge, even at application level there are **no readings considering integration** components and mechanisms to connect to other systems. Publications are however more present for the data layer; nevertheless, papers similarly focus on customers' data during application execution leaving aside the data model used by system administrators.

3 Goals

The main objective of this this thesis is to define an approach that fosters **component reuse** and extends traditional MT architectures so that **multiple functionalities can be served using the same software instance**.

This unification aims to reduce costs and widen the spectrum of clients to software providers by adding varied and enlarged markets; the approach also intends to help programmers to find support for agility in development (decreasing time-to-market) as well as improve the agility in deployment (Figure 3).

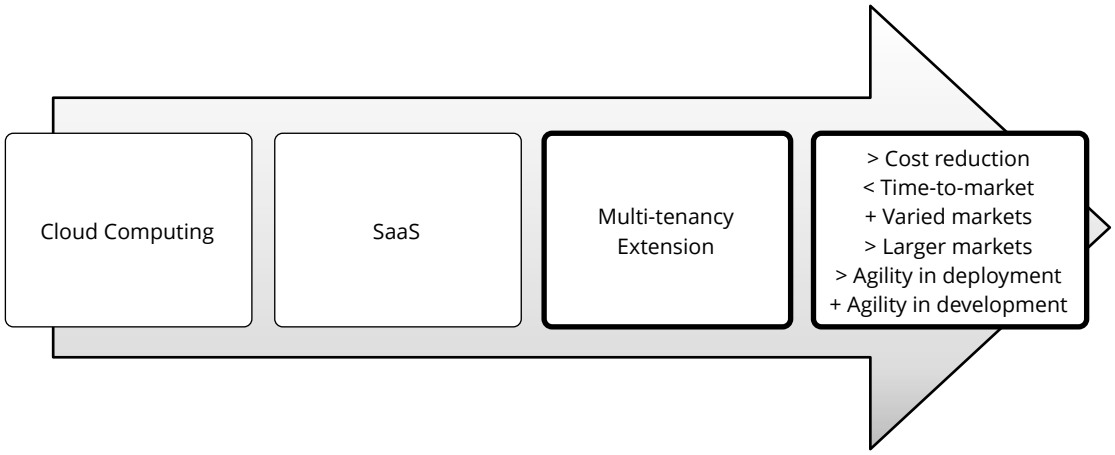


Figure 3. Steps to benefits in SaaS MT extended.

In order to accomplish this main objective, the following intermediate goals have been also identified for the success of this research work:

- Devise and define a high-level architectural framework for MTAs at both administrative and instance (application) level; define tiers, components, relations and integration mechanisms with other applications. This contribution is, to the best of our knowledge, the first explicitation of such an architectural pattern.

- Detect and point out deficiencies and inconveniences in current multi-tenancy. Weigh out possible solutions, balance modifications and fix reusability as the cornerstone of our research.

- Analyze the types of EIS thoroughly and raise MTAs to a multi-functional level identifying components that could be divided into reusable/common and non-reusable/individual assets.

- Determine required administrative features to control and support multi-functional MT environments. Analyze functional management, functional scalability, multi-functional contracts maintenance and system balance.

- Study mechanisms to perform multi-functional deployment in MT applications. Combine methodologies from traditional multi-user environments with the new multi-functional setting.

- Find out new security issues related to the approach; define levels of privacy and methods to tackle the new circumstances.

- Provide a general model for extended MTAs. Use the architectural framework proposed as the base for the extension. Describe the new elements with its categorization, roles and interrelations. Consider all levels of the architecture and include integration mechanisms.

- Define a reference framework for the design of MT databases. Based on this contribution, detail a relational database model to support multi-functional MT applications. Illustrate entities and relations, explain functional scalability support and articulate levels of privacy.

- Identify a methodology to determine and incorporate common reusable components into extended systems. Find out how to make use of non-common assets without system overhead and security breaches.

- Frame the new architecture into the current trends that try to align agility and architecture. Elicit the benefits that traditional and extended MTAs provide in this regard.

- Illustrate how the approach fosters the SaaS embracement by SMEs, both for customers and vendors. Clarify how this extension makes useful and effective applications more appropriate for this range of companies.

- Show the applicability of the approach by implementing a complex software system with core extended MT architecture. Probe the benefits and discover shortcomings/opportunities of the extension proposed.

4 Research methodology

The steps followed during this research can be summarized as follows:

- Investigate the origins of multi-tenancy. Deepen into the insights of cloud computing and understand MTAs as a fundamental part for the success of SaaS as a new software distribution formula.

- Study profoundly the state of art of MTAs. Elicit an architectural framework for MT applications devising their constituents and component design.

- Find out the shortcomings of current multi-tenancy as a mono-functional environment. Discuss how this situation affects vendors, companies and programmers. Explain how these pitfalls can be overcome by using multi-functional applications.

- Design a new extended MTA model in order to solve the deficiencies and boost component sharing. Define new components, purpose and topological relationships. Specify modifications of the extension at all levels of the architecture.

- Fully review the state-of-art for multi-tenant databases. Understand data privacy as the major concern for the use of shared environments. Explain approaches for implementing multi-tenancy and evaluate pros and cons of the different schema mapping techniques for customization.

- Define a reference framework for the design of MTAs' data layer. Use it as a basis for illustrating a relational database model for our approach.

- Understand the Agile Software Development (ASD) philosophy and its methodologies. Consider the current trend to align agility and architecture. Discuss agility in SaaS multi-tenant applications, its differences to ASD, and its subsequent challenges. Study the advantages that this novel extension entails on the field of agile architectures.

- Research on EIS studying the origin, types and advantages of its implementation. Analyze factors for companies to move to the cloud and migrate legacy systems to SaaS enterprise solutions. Consider and pay special attention to SMEs as central players in the world's economy.

- Design and implementation of *Globalgest,* a real application based on the new proposed MT architecture. Establish a methodology in this development for the reutilization of common components among functionalities. Discuss practical shortcomings and possible future improvements.

- Carry out experiments to demonstrate the utilization of Globalgest by companies from mixed industries and fitting in the range of SMEs.

5 Structure of the thesis

The thesis is divided in 4 main parts and comprises a total of 11 chapters (Figure 4). After this first introduction part, Part II studies and analyzes the current theoretical framework for the problem statement purpose, Part III describes the proposal, and Part IV presents a summary of conclusions and future work.

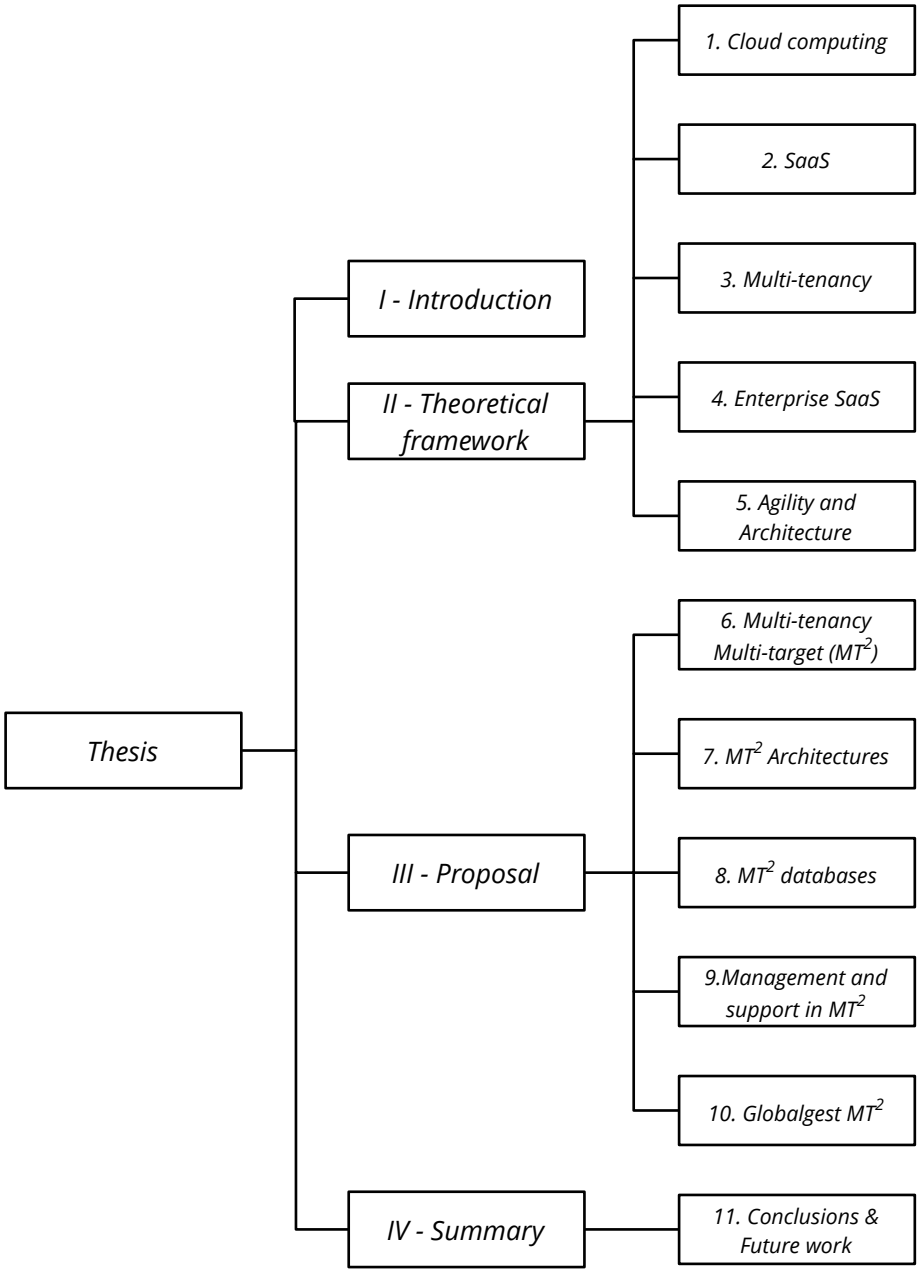


Figure 4. Structure of the Thesis

In this research work, chapters follow a logical order with the intention to clarify the foundations and place the proposal within the context of an extension to the traditional multi-tenant model:

- Chapter 1 reviews cloud computing carefully, studying the origins, levels of service, benefits as well as obstacles and opportunities.

- Chapter 2 focuses on SaaS as the cloud computing software distribution formula. This section studies its beginnings, types, key aspects, benefits and disadvantages.

- Chapter 3 studies multi-tenancy in depth. We analyze the state of art of MTAs and MT databases explaining the architecture, ways of implementing multi-tenancy and schema mapping techniques for system personalization.

- Chapter 4 introduces the concept EIS explaining some of the best-known types of applications. This chapter also reviews the penetration of EIS in the world of SMEs as well as factors that affect its cloud migration and embracement.

- Chapter 5 summarizes the agility movement, its beginnings and methodologies. Simultaneously, this chapter studies the actual tendency for the cohabitation of agility and architecture. Lastly, chapter narrows down to the benefits of MTAs in terms of agility.

- Chapter 6 presents the MT^2 (after Multi-Tenancy Multi-Target), as the next step to MT. We review the motivation and illustrate the proposal in different scenarios that clarify the foundations and benefits for this novel approach. This chapter also contrasts the advantages of the proposal in terms of agility and analyses the suitability to SMEs.

- Chapter 7 proposes a high-level architectural framework for MTAs. After setting this contribution as a base for our extension, a component-based design of MT^2 architectures is fully explained and depicted.

- Chapter 8 focuses on the data layer of MT^2As. Firstly, it proposes a reference model for the design of MT databases; secondly, it provides a detailed relational model for MT^2 data layer. It is important to emphasize that an extensibility pattern has been described.

- Chapter 9 explains tools and mechanism to support and manage MT^2 applications. Critical aspects of extended systems such as system balance, functional management or multi-service subscriptions will be considered.

- Chapter 10 demonstrates the applicability of the approach by presenting Globalgest, a real commercial enterprise software solution, which is an implementation of the proposal. This software is currently in production and deploys 225 functionalities in an integrated way serving more than 40 companies from 17 heterogeneous industries.

- Conclusions and ongoing work are detailed on chapter 11.

As general recommendation for readers who can be very familiarized with the context of this research work (i.e. Cloud Computing, Multi-tenant Architectures, Enterprise Information Systems, and Agility), and due to the extension of the study and analysis presented in Part II, perhaps they can put the focus only on Part III and IV of this document.

Capítulo I

Cloud computing

Este primer capítulo introduce el concepto de cloud computing. Se mostrarán sus orígenes y la razón de su desarrollo actual así como varias definiciones. Se presentan también los tipos de servicios cloud según diversas clasificaciones. Finalmente, se analizan limitaciones y nuevas oportunidades de investigación.

1 Introducción

Cloud Computing (computación en la nube) es el paradigma soñado desde hace décadas para tener computación como servicio (*Utility Computing*) [30]. Cloud computing tiene la capacidad de cambiar gran parte de la industria TIC [4] (Tecnologías de la Información y de la Comunicación, en inglés *Information Technology*, IT) y supone no sólo un cambio en el concepto de distribución del software (que pasa a ser un servicio - *Software as a Service*), sino que también elimina las barreras de diseño en la creación de nuevos productos. Los programadores no ven truncadas sus ideas por falta de capital ya que los recursos pueden ser asignados de forma flexible [4], [31] y sin necesidad de afrontar inversiones iniciales. De este modo, los desarrolladores parten desde la misma línea de salida y muchos más pueden alcanzar sus objetivos de creación de nuevos productos competentes.

Hardware, ancho de banda y software operacional de la última generación; todos ellos pueden ser contratados bajo demanda por cualquier empresa o particular. Gracias a la nube, complejidades computacionales imposibles se realizan ahora a velocidad vertiginosa. Tenemos miles de servidores a nuestra disposición, elasticidad de recursos, ubicuidad en el acceso a las aplicaciones, plataformas de desarrollo y multi-tenancy para economías en escala.

Cloud computing supone una "*situación sin precedentes en las Tecnologías de la Información y la Comunicación"* [4].

La Figura 5 compara el paradigma anterior con esta nueva situación. En *in-house computing*, la capacidad computacional permanece en las propias instalaciones de las empresas. En este caso es necesario tanto una inversión inicial en infraestructura así como un personal de mantenimiento informático cualificado. En cambio, cloud computing propone un modelo de migración de capacidades a un tercero que es el que realiza la inversión inicial y mantiene la infraestructura y el personal. Estas empresas *proveedoras cloud* cobran por este servicio en diferentes niveles, dependiendo de nuestras necesidades. De este modo, las organizaciones optan por primera vez a recursos computacionales de última generación, hasta entonces reservados para las grandes corporaciones.

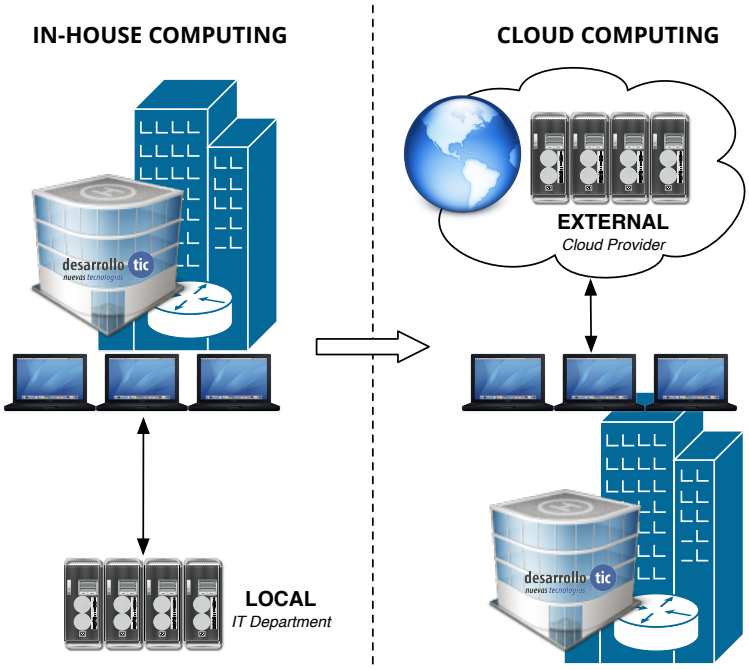


Figura 5. In-house computing vs. cloud computing

2 Orígenes del Cloud Computing

Cloud Computing es un nuevo término usado para referirse a una idea antigua de los años 60: *Utility Computing* (computación como servicio/utilidad) [1], [30], [32]. Esto es, el **suministro de servicios computacionales bajo demanda**. Hoy en día, la percepción a considerar el futuro de la computación como la 5ª utilidad está más extendida [1]–[4]. Del mismo modo que pagamos teléfono, gas, agua y electricidad, pasaremos a pagar la computación consumida. Sin embargo, el camino hasta aquí no ha sido fácil.

John McCarthy fue el primero en visionar este paradigma cuando en el año 1961. Durante la conferencia en el MIT Centennial, *"Architects of the Information Society"* McCarthy dijo [32]:

"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility…The computer utility could become the basis of a new and important industry."

El término "cloud" también ha sido usado en otros contextos, como por ejemplo para describir las redes ATM (*Asynchronous Transfer Mode*) durante los años 90. Sin

embargo, no empezó a ganar popularidad hasta que el CEO (Chief Executive Officer) de Google Eric Schmidt en el año 2006 habló de un nuevo modelo de negocio para suministrar servicios vía Internet. Hasta entonces, el vocablo cloud computing había sido usado fundamentalmente como un activo de marketing en diferentes contextos [1].

Sobre cloud computing se ha hablado, escrito y citado mucho en congresos, conferencias y revistas. La confusión ha llegado a reinar incluso en la definición exacta [4] y de hecho, provocó hasta frustraciones, tal y como le ocurrió a *Larry Ellison*, máximo responsable de Oracle:

"The interesting thing about cloud computing is that we've redefined cloud computing to include everything that we already do... I don't understand what we would do differently in the light of Cloud Computing other than change the wording of some of our ads. " Larry Ellison, quoted in the Wall Street Journal, September 26, 2008

También encontramos declaraciones del vicepresidente de ventas de HP en Europa:

A lot of people are jumping on the [cloud] bandwagon, but I have not heard two people say the same thing about it. There are multiple definitions out there of "the cloud." Andy Isherwood, quoted in ZDnet News, December 11, 2008

Richard Stallman, creador del sistema operativo GNU y fundador del la *"Fundación del Software Libre"* describió cloud computing como una "trampa" y una "estrategia de marketing" [26]. El mismo Larry Ellison también consideró a cloud computing como un "Galimatías" del que sería muy difícil hacer dinero al carecer "claramente de un modelo de negocio" [26].

Como podemos ver, incluso los más grandes han llegado a estar confusos con la definición. Como consecuencia, este desconcierto provocó desengaños y ocasionó el nacimiento de detractores que no veían futuro en esta tecnología.

Por esta razón, los investigadores se han volcado para intentar estandarizar la definición de cloud computing. Aunque todavía no existe un consenso completo, si que hay posturas semejantes e incluso definiciones aceptadas. En las siguiente sección profundizaremos en este asunto.

Cloud computing está llamado a causar el mismo impacto en el software que el que han causado las industrias metalúrgicas en el hardware. Existen similitudes entre ambas circunstancias que pasamos a analizar.

Hoy en día, tener una línea de producción propia de hardware, está reservado a las grandes corporaciones. El coste de adquisición de una línea de fabricación de semiconductores puede alcanzar los tres billones de dólares. Debido a ello, solamente grandes empresas como Intel o Samsung pueden disponer de su propia fundición. A raíz de este problema, han surgido empresas que fabrican chips a terceros como la *Taiwán Semiconductor Manufacturing Company* (TSMC). De esta forma, las empresas centran sus esfuerzos en el diseño de chips innovadores, delegando la producción de los mismos en otras.

Gracias a esto, empresas como *nVidia* pueden convertirse en un referente en el sector, sin tener los costes operacionales, gastos de inversión y el riesgo que supone poseer una línea propia de producción. Por otro lado, las empresas fabricantes de chips mejoran notablemente el rendimiento de su línea de producción al existir una gran cantidad de empresas que carecen de fundición pudiendo multiplexar el uso de la misma [4].

La virtualización ha sido la técnica clave para posibilitar esta visión de la computación como otro utilidad [33], ya que ha posibilitado la creación de datacenters a gran escala que ofrezcan a las empresas su uso bajo demanda. Si aplicamos esta analogía al desarrollo de software y sus necesidades de computación cada vez más costosas, entenderemos el amplio salto de calidad que supone la externalización de los recursos computacionales como un servicio, al disminuir el coste en adquisición de recursos para la producción del software.

3 Definición de cloud computing

Todavía no existe un común acuerdo para enunciar el concepto de cloud computing [3], [14], [26], [34]–[37]. Existen varias definiciones que, aunque parecidas presentan diferencias. De todas ellas, la del Instituto Nacional de Estándares y Tecnologías[1] (*NIST- National Institute of Standards and Technolog*y) es la que ha tomado más fuerza

[1] *Entidad perteneciente al Departamento de Comercio del Gobierno Federal de los Estados Unidos.*

y se suele tomar como base. Además de esta, también está presente la acepción de la Universidad de Berkeley, que también se analiza por ser una de las más referenciadas.

3.1 Cloud computing según NIST

En su publicación *"The NIST Definition of Cloud Computing"* [33], esta institución define cloud computing como:

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"

Según esta definición, cloud computing es un modelo que permite el acceso en red y bajo demanda a un conjunto compartido de recursos computacionales configurables (redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente asignados y liberados con un esfuerzo mínimo de gestión e interacción del proveedor. Según [3], con esta definición NIST pretende subrayar que cloud computing no es una tecnología nueva, sino que más bien un nuevo modelo para la distribución de infraestructura, servicios e información computacional que uso de tecnologías que son explotadas y ofrecidas por los proveedores de servicios cloud.

Hay **cinco características esenciales** que definen a cloud computing [33], [38]:

1. *Autoservicio bajo demanda*: el usuario puede consumir recursos (como procesamiento o almacenamiento) a medida que los va necesitando, sin pedirlo expresamente al proveedor del servicio.

2. *Acceso a través de red*: Los recursos estás disponibles a través de la red y se accede a ellos mediante mecanismos estándar que promueven el uso de diferentes plataformas cliente (e.g., teléfonos móviles, tablets, portátiles y estaciones de trabajo).

3. *Compartición de recursos*: Los recursos computacionales del proveedor son agrupados y distribuidos entre los diferentes usuarios siguiendo un modelo *multi-tenant*. De este modo los diferentes recursos (virtuales o físicos) son asignados y

liberados en función de la demanda de los clientes. Existe una impresión de transparencia en cuanto a la localización de recursos en el sentido de que los clientes normalmente no saben donde están ubicados. Sin embargo, en ocasiones los clientes deben ser capaces de especificar la localización en un nivel mayor de abstracción (latencia o por motivos legales, ya que determinados países obligan a que estén dentro de sus fronteras [14], [39]). Ejemplos de recursos pueden ser: almacenamiento, procesador, memoria y ancho de banda.

4. *Rápida elasticidad*: los recursos pueden ser suministrados y liberados de forma elástica. En ocasiones de forma automática por motivos de escalado para ser capaces de satisfacer la demanda. El cliente tiene la impresión de estar ante un conjunto de recursos ilimitado de los que puede hacer uso en cualquier momento, y cualquier cantidad.

5. *Servicio medible*: Los sistemas cloud son capaces de optimizar el uso de recursos aprovechando las diferentes métricas de consumo de los mismos. Los recursos pueden ser monitorizados, controlados y comunicados brindando transparencia, tanto al cliente como al proveedor de servicio.

Los diferentes **niveles o servicios** en los que se divide cloud computing son:

- **Software as a Service (SaaS)**: Acceso a aplicaciones software.
- **Platform as a Service (PaaS)**: Acceso a plataformas de desarrollo.
- **Infrastructure as a Service (IaaS)**: Acceso a Infraestructura hardware.

Estos servicios (que serán analizados en detalle posteriormente) pueden ser desplegados en cuatro **diferentes modelos de despliegue**:

- **Nube privada** (*Private Cloud*): La infraestructura es operada por una sola organización.

- **Nube Comunitaria** (*Community Cloud*): La infraestructura es operada por varias organizaciones con intereses y fines comunes. Pueden ser gestionadas por la misma comunidad o un tercero.

- **Nube pública** (*Public Cloud*): La infraestructura está disponible para todo el público general o una gran industria. Está gestionada por un proveedor que comercializa sus servicios.

- **Nube híbrida** (*Hybrid Cloud*): La infraestructura cloud está compuesta por una o varias nubes (privadas, comunitarias o públicas), que permanecen como entidades únicas pero que están ligadas por una tecnología estándar o propietaria que permite migración de datos y aplicaciones (por ejemplo para el balanceo de carga entre nueves)

Esta acepción de NIST suele ser la más referenciada y en los trabajos más contemporáneos parece estar consolidada. **Por ser la más extendida, es la que se usará en este trabajo**. De este modo, el resto de secciones en las que se extiende el concepto, la tomarán como base y punto de partida.

3.2 Cloud computing según Berkeley

La Universidad de Berkeley en su artículo "*Above the Clouds: A Berkeley View of Cloud Computing*" [4] comenzó a clarificar el paradigma:

"Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS). The datacenter hardware and software is what we will call a Cloud. We use the term Private Cloud to refer to internal datacenters of a business or other organization, not made available to the general public. Thus, Cloud computing is the sum of SaaS and Utility Computing, but does not include Private Clouds"

Según lo anterior, cloud computing comprende tanto las aplicaciones SaaS, como los centros de datos (datacenters) que proporcionan el hardware y el software necesario para alojar dichas aplicaciones. El conjunto de dichos centros de datos es conocido con el nombre de *nube* (cloud). Cuando la nube se comercializa para el público general en un régimen de pago por uso, se le conoce con el nombre de *nube pública* (public cloud en inglés) y el servicio que se comercializa *Utility Computing*. Por el contrario, se le conoce con el nombre *nube privada* (private cloud) cuando los datacenters pertenecen a

una organización privada y no son accesibles al público en general. Por tanto, según Berkeley cloud computing es la suma de SaaS y Utility Computing, pero sin incluir las nubes privadas [4] (Figura 6).

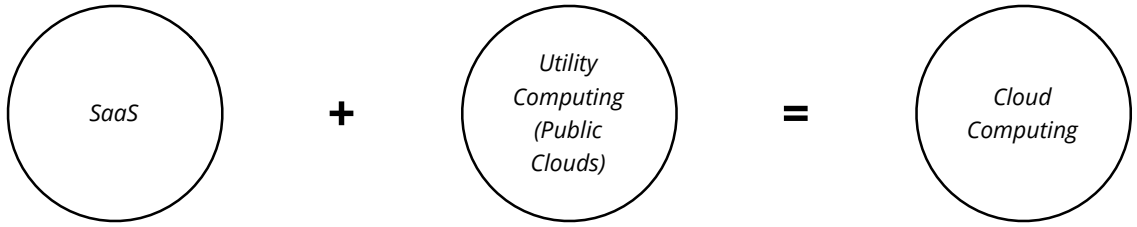


Figura 6. Cloud computing según Berkeley es el despliegue de SaaS en nubes públicas

En esta visión, diferentes niveles de utility computing serán ofrecidos en función del nivel de abstracción que se le presente al programador y de la capacidad de acceso a la gestión y configuración de recursos.

Desde el punto de vista hardware, cloud computing supone tres nuevas ventajas principales [40]:

1. La impresión de tener **recursos computacionales infinitos**, según nuestras necesidades. Eliminamos por tanto, una previsión planificada de los mismos. Ya que, a medida que los vayamos necesitando, los podremos ir demandando.

2. No es necesario un compromiso por adelantado de recursos para el consumidor cloud. Una empresa puede **empezar con escasos recursos** e ir aumentándolos a medida que los necesite.

3. Posibilidad de **pago en pequeñas fracciones según el uso** (por ejemplo, el pagar por horas los procesadores o diariamente el almacenamiento) e ir asignando más a medida que sea necesario. Evitamos así el sobre aprovisionamiento y optimizamos el uso de las máquinas. De esta forma, favorecemos a los usuarios por un lado, ya que tienen acceso a una tecnología cuya compra sería inalcanzable y a los vendedores por otro, maximizando el uso de sus recursos.

3.3 Otras definiciones

Existe otra definición de cloud computing que también podemos considerar representativa y que incluye aspectos como la *Calidad del Servicio* (QoS, Quality of Service,) y la *computación pervasiva* [34]:

"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".

En [41] el autor compara más de 20 definiciones de cloud computing para intentar obtener un estándar. Finalmente propone la siguiente definición en la que se tienen en cuenta aspectos como virtualización y los *Acuerdos a Nivel de Servicio (*SLAs, Service Level Agreement*):*

"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs".

En [42], Pyke intenta acercar el concepto de cloud computing para hacerlo más entendible a la gente de a pie, con pocos conocimientos tecnológicos y computacionales:

"Para los geeks, cloud computing has sido sinónimo de grid computing, utility computing, software as a service, virtualización, aplicaciones basadas en web, computación autónoma, computación punto a punto y procesamiento remoto - así como de combinaciones de los términos anteriores. Para los no geeks, cloud computing es simplemente es una plataforma donde individuales y organizaciones utilizan Internet para acceder a recursos software y hardware interminables que cubren la mayoría de sus necesidades, dejando a empresas terceras las complicaciones informáticas".

Como resumen podríamos concluir que cloud computing significa el uso de recursos computacionales bajo demanda a diferentes niveles y a través de un *Cloud* (Nube) [14].

Un cloud es un centro de datos que ofrece infraestructura computacional (hardware o software) y al cual se accede a través de Internet. De esta forma, las empresas son capaces de migrar su infraestructura TIC reduciendo o eliminando los costes de inversión inicial y operacionales. Además, las organizaciones se ven beneficiadas al trasladar las complicaciones técnicas a empresas terceras, que son las proveedoras cloud.

4 Roles y actores en cloud computing

En cloud computing tenemos tres tipos de rol o papel dependiendo del nivel de servicio. Los diferentes actores los podemos clasificar en: empresas proveedoras de infraestructura (más dedicadas al hardware), empresas desarrolladoras y/o comercializadoras de software y por último los usuarios de estas aplicaciones [31].

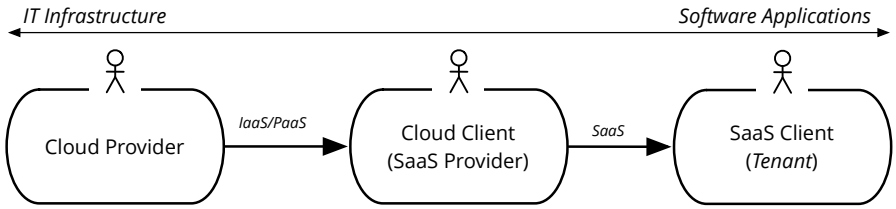


Figura 7. Actores en cloud computing

La Figura 7 ilustra los diferentes actores existentes así como la forma en la que acceden y participan en el paradigma:

- ***Cloud Provider***: Grandes compañías poseedoras de datacenters que ofertan como servicio el consumo de capacidad computacional y recursos hardware según demanda (IaaS) así como de plataformas de desarrollo (PaaS).

- ***SaaS Provider***: Desarrolladores de software que despliegan sus aplicaciones en la nube haciendo uso de los datacenters de los *Cloud Providers*. Dichas aplicaciones son comercializadas como SaaS, desapareciendo el concepto de licencia de pago único y pasando al concepto de pago por uso, mensualmente, anualmente, etc.[43]

- ***SaaS User***: Clientes que abonan las licencias para el uso de las aplicaciones SaaS. No tienen porqué conocer detalles sobre la plataforma en la que las aplicaciones se encuentran alojadas.

Los usuarios y los roles no son excluyentes, sino que puede que existan proveedores cloud que a su vez desplieguen aplicaciones SaaS en su propia nube o en otra para comercializarlas.

Existen autores [1], [4] que usan el término *utility computing* para describir el servicio ofertado por los cloud providers. En nuestro caso, siendo fieles a los niveles de servicio de la definición de NIST, los denominaremos IaaS y PaaS.

5 Servicios cloud computing

En esta sección se detallan los principales servicios ofertados en la nube. En primer lugar y siguiendo la línea de esta tesis detallaremos los servicios considerados en el trabajo de NIST [33]; posteriormente, presentaremos otros tipos y definiciones también existentes en la literatura [34], [44] destacando la categorización realizada por la Universidad de Berkeley [4], en la que se tiene en cuenta una división continua por el nivel de abstracción y acceso a la capa hardware.

5.1 Principales modelos de servicio en la nube

Definimos *infraestructura cloud* como una colección de hardware y software que posibilita las cinco características esenciales de cloud computing (previamente estudiadas). La infraestructura cloud puede estar constituida por una *capa física* y una *capa abstracta*. La capa física consiste en todos aquellos recursos hardware necesarios para dar soporte a los servicios cloud ofrecidos, que típicamente son servidor, almacenamiento y componentes de red. La capa de abstracción está formada por el software desplegado en la capa física y que pone de manifiesto las cinco características esenciales. Conceptualmente, la capa de abstracción está situada por encima de la capa física.

La Figura 7 nos muestra los tres niveles principales de acceso a la nube. Tal y como podemos observar, en los niveles inferiores nos acercamos a un control y gestión de la infraestructura cloud mientras que los niveles superiores están destinados más al uso de aplicaciones. En las siguientes subsecciones pasamos a detallar cada uno de ellos. Esta misma clasificación también está presente en [41], sin embargo ofrecemos las definiciones de NIST por considerarlas más acertadas.

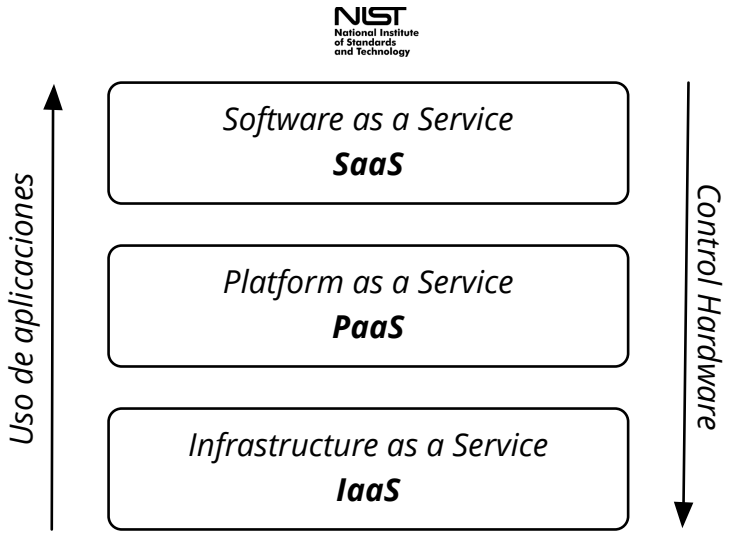


Figura 8. Modelos de servicio en la nube según la clasificación NIST.

5.1.1 Infrastructure as a Service (IaaS)

En este caso, el servicio proporcionado al cliente consiste en la provisión de almacenamiento, red y otros recursos computacionales básicos. El cliente es capaz de instalar y ejecutar software de forma arbitraria, este puede consistir bien en sistemas operativos o en aplicaciones. El cliente no tiene capacidad para gestionar la *infraestructura cloud* subyacente pero si que tiene control sobre los sistemas operativos, el almacenamiento y aplicaciones instaladas; posiblemente también tenga una control limitado para la selección de determinados componentes de red (e.g., firewalls) [33].

En IaaS, los recursos tradicionales de computación como servidores, almacenamiento o red, se ofrecen de manera virtual en Internet y bajo demanda. En la literatura nos podemos encontrar con términos sinónimos como *Hardware as a Service (HaaS)* [5], [37]. HaaS fue acuñado en el año 2006 como resultado de los grandes avances en la virtualización de hardware, automatización y métricas de uso de recursos [34], [36]. Los usuarios podían adquirir hardware, incluso un datacenter completo, siguiendo un régimen de pago por demanda.

Este tipo de servicio es extremadamente útil para las empresas ya que **elimina la necesidad de una inversión inicial** para la creación y gestión de una infraestructura TIC propia [5].

Otra ventaja importante es la posibilidad de acceder a la última tecnología a medida que ésta aparece. Las empresas pueden actualizar su infraestructura IT de forma casi inmediata.

Figura 9. Amazon Web Services. Líder en IaaS

Los ejemplos de este tipo de comercialización más famosos son Amazon Elastic Compute Cloud (EC2) [45], Google Compute Engine [46], Windows Azure [47] o Rackspace [48]. De todas ellas, la primera empresa en ofrecer servicios de cloud computing fue Amazon a través de su plataforma Amazon web services (Figura 9); según un informe de Gartner [49] todavía se mantiene líder. En el caso de Microsoft, su plataforma fue lanzada inicialmente como PaaS pero posteriormente Microsoft añadió el nivel más bajo de abstracción y control con el objetivo de hacerla más atrayente. Como caso especial, destacamos otros productos OVH Dedicated Cloud [50] que te permite contratar un datacenter virtual con capacidad de hasta 10.000 máquinas virtuales (MVs) en menos de 30 minutos.

5.1.2 Platform as a Service (PaaS)

Con este servicio, los clientes son capaces de instalar en la infraestructura cloud aplicaciones propias o de terceros. Estas aplicaciones son creadas usando lenguajes de programación, librerías, servicios y herramientas ofrecidas por el proveedor cloud. El cliente no gestiona ni controla la infraestructura cloud subyacente incluyendo redes, servidores, sistemas operativos o almacenamiento. Por el contrario si tiene control sobre las aplicaciones instaladas y posiblemente sobre opciones de configuración del entorno sobre el que se alojan dichas aplicaciones.

Como su propio nombre indica, lo que PaaS ofrece a los programadores es una plataforma con un entorno pre configurado proporcionando todo lo que necesitan para el desarrollo y que les permite olvidarse de complejas instalaciones iniciales del

software (Apache, PHP, Java Runtime, etc.). Por tanto, las empresas desarrolladoras se dedican al código desde el primer minuto.

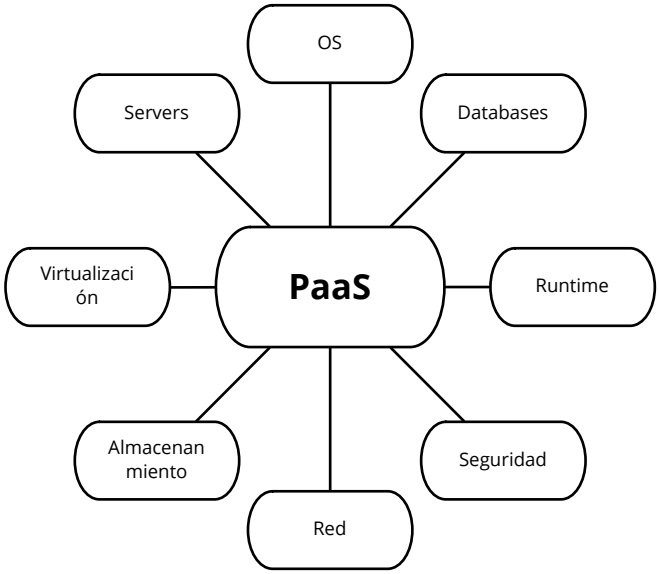


Figura 10. Platform as a Service proporciona abstracción a los programadores

Por tanto, PaaS es una plataforma de programación en la nube en la que los programadores no se tienen que preocupar del sistema operativo, almacenamiento, alojamiento y otros elementos pertenecientes a capas inferiores (ver Figura 10). Los desarrolladores pueden trabajar de forma local en sus ordenadores y les basta con presionar una sola tecla para que el código sea desplegado a la plataforma y esté disponible en Internet en cuestión de segundos.

El proveedor PaaS se encarga del mantenimiento, seguridad y actualización de la plataforma, en definitiva, de todos los aspectos relacionados con el buen funcionamiento de la aplicación. Normalmente, los proveedores PaaS tienen limitada la plataforma en cuanto al uso de determinados lenguajes de programación (por ejemplo python para el caso de Google App Engine [51]). Además, en la mayoría de los casos los clientes PaaS no pueden acceder a bajo nivel al sistema operativo y las aplicaciones que funcionan bajo una plataforma PaaS suelen tener unas consideraciones mayores en seguridad para evitar accesos no deseados.

Otro ejemplo de proveedor PaaS, además de la ya citada Google App Engine, es Force.com [52], que es utilizado por SalesForce.com [53] (un proveedor SaaS de CRM). Elastic Beanstalk es el producto PaaS de la suite AWS (Amazon Web Services);

este producto despliega ya en su entorno HTTP Apache para Node.js, PHP, Python, Passenger para Ruby, IIS 7.5 para.NET, Apache Tomcat para Java y Docker.

5.1.3 Software as a Service (SaaS)

El servicio ofrecido los usuarios es el uso de aplicaciones de un proveedor, ejecutadas en una infraestructura cloud. A las aplicaciones se puede acceder bien a través de una interfaz de un clientes ligero (*thin clients*), como un navegador web (e.g, email basado en web) o a través de la interfaz de un programa. El cliente no gestiona ni controla la infraestructura cloud subyacente incluyendo redes, servidores, sistemas operativos o almacenamiento e incluso las mismas aplicaciones, con las excepción de un limitado número de opciones de configuración a nivel de usuario final.

El software y las aplicaciones se alojan como servicios y los usuarios acceden a ellas a través de Internet. De este modo, los usuarios evitan tener que instalar las aplicaciones localmente para poder ejecutarlas. A su vez, además de la disminución del precio, el usuario ya no tiene la ardua tarea de actualización y mantenimiento, recayendo esta responsabilidad sobre el proveedor de SaaS.

Las ventajas del SaaS, tanto para los desarrolladores, como para los usuarios finales son fáciles de entender. Por un lado, se simplifica la instalación y el mantenimiento y por otro, los usuarios finales pueden disfrutar de la aplicación 24h-365 días al año de manera pervasiva, centrándose en su trabajo y sin preocuparse de actualizaciones, seguridad, etc.

De cara al consumidor de SaaS, cloud computing es totalmente transparente y para él no varía apenas la situación. La principal diferencia existe para el proveedor SaaS que ve disminuidos sus costes de infraestructura al no necesitar poseer un data center. Por otro lado, los proveedores cloud ven optimizado el rendimiento de sus centros de datos al poder distribuir el uso entre diferentes clientes. Estamos de nuevo en la analogía explicada anteriormente con las empresas productoras de semiconductores que dan la oportunidad a empresas pequeñas de diseñar y vender chips sin necesidad de disponer de una fábrica propia.

Cloud computing permite desplegar aplicaciones SaaS a empresas desarrolladoras de software sin necesidad de acometer gastos iniciales de inversión y provisión. Además,

las necesidades de capacidad computacional pueden ser escaladas hacia arriba o abajo dependiendo de la demanda. Es importante tener en cuenta que determinados riesgos procedentes del soporte tales como seguridad y fiabilidad son delegados al proveedor cloud.

Cloud computing **iguala la línea de salida** en la carrera SaaS, en el sentido de permitir que todos los desarrolladores puedan disfrutar de las mismas oportunidades sin depender de factores económicos.

5.2 Clasificación continua de cloud computing.

A pesar de que la anterior clasificación es la más extendida y consensuada. Merece la pena presentar la división de servicios establecida por la Universidad de Berkeley [4]. En ella se establece una distribución continua en función del nivel de abstracción proporcionado al programador.

Toda aplicación distribuida compleja precisa de un modelo de computación, un modelo de almacenamiento y un modelo de comunicación. La multiplexación necesaria para alcanzar la elasticidad y la impresión de capacidad infinita de computación requiere la virtualización de recursos. De este modo, los detalles acerca de cómo los recursos son compartidos se ocultan al programador que permanece en un nivel de abstracción superior. El espectro de separación de los diferentes niveles en los que cloud computing se ofrece es continuo, los proveedores no tienen porqué pertenecer a un tipo u otro, sino que pueden encontrarse en la frontera de ambos. En la Figura 11 preciamos una representación del espectro en la que están representados servicios de distintos proveedores que analizamos a continuación.

Figura 11. Clases de cloud computing según el nivel de abstracción al programador

Amazon EC2 [45] se encuentra en uno de los extremos. Sus siglas quieren decir *Elastic cloud computing* y se trata de un servicio que proporciona capacidad de infraestructura informática bajo demanda. Los usuarios pueden contratar instancias de máquinas

virtuales con diferentes configuraciones según sus necesidades. En la Figura 12 podemos ver una captura de la consola de administración para los usuarios.

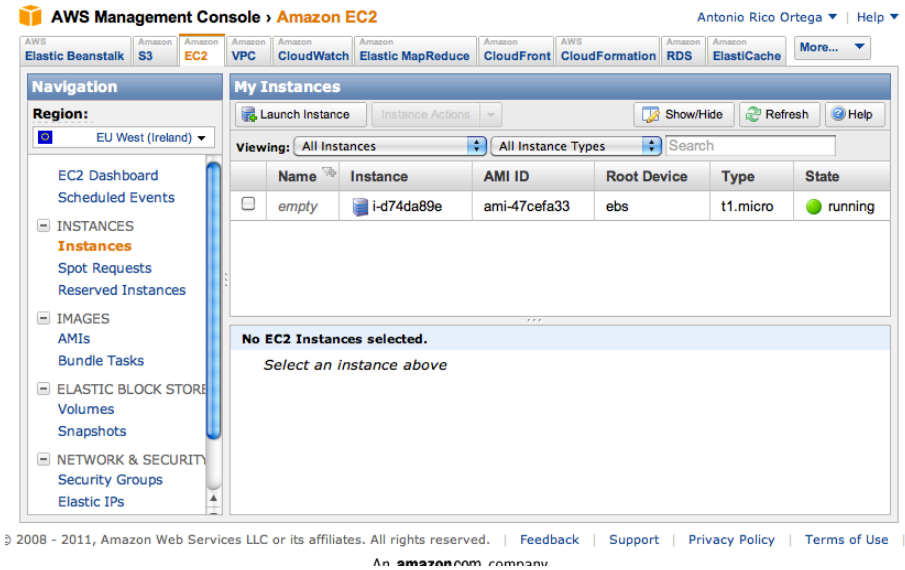


Figura 12. Consola de administración Amazon Web Services. Servicio EC2

Una instancia de EC2 se parece mucho a un elemento hardware y los usuarios pueden controlar casi todo el software al más bajo nivel. La API (Application Programming Interface) que se proporciona permite en unas pocas llamadas configurar la virtualización de hardware. No existe limitación del tipo de aplicaciones que se pueden alojar y los programadores pueden escribir el código que quieran. Por el contrario, esta versatilidad prácticamente imposibilita a Amazon ofrecer escalabilidad y robustez, al estar muy ligadas a la aplicación.

En el otro lado del espectro nos encontramos con plataformas como *Google App Engine* [51] *y Force.com* [52]. *AppEngine* está destinada a aplicaciones web tradicionales, forzando a las aplicaciones a estar basadas en un protocolo de consulta-respuesta y por tanto, a tener un racionamiento severo en la cantidad de computación que puedan usar para dar una determinada respuesta. Por el contrario, presenta un mecanismo automático de escalabilidad y robustez, que permite la elasticidad en los recursos, pudiendo consumirlos según se necesiten. Force.com está diseñado para alojar aplicaciones que trabajen con la base de datos de *SalesForce.com* [53] y solamente con ella.

Microsoft Azure se encuentra en el punto medio del espectro. Busca el equilibrio entre la flexibilidad y la conveniencia del programador. Las aplicaciones Azure se escriben

usando bibliotecas .NET y compiladas a *Common Language Runtime*, un lenguaje independiente del entorno. Soporta todo tipo de aplicaciones y los usuarios pueden escoger el lenguaje. Sin embargo, no pueden gestionar el sistema operativo subyacente ni las rutinas. Las bibliotecas permiten un cierto grado de configuración de red y escalabilidad, pero es necesario que el programador realice ciertas declaraciones en el código durante el desarrollo. Por tanto, Azure supone un punto medio entre frameworks de desarrollo de aplicaciones como AppEngine y máquinas hardware virtuales como las EC2 de Amazon WS.

Si comparamos la división de NIST y la de Berkeley podemos ver como en realidad **se trata de una misma división** (Figura 13). El extremo donde se ubica AWS se correspondería con IaaS, la zona central del continuo equivale a los servicios PaaS mientras en el extremo restante tiene su análogo en SaaS.

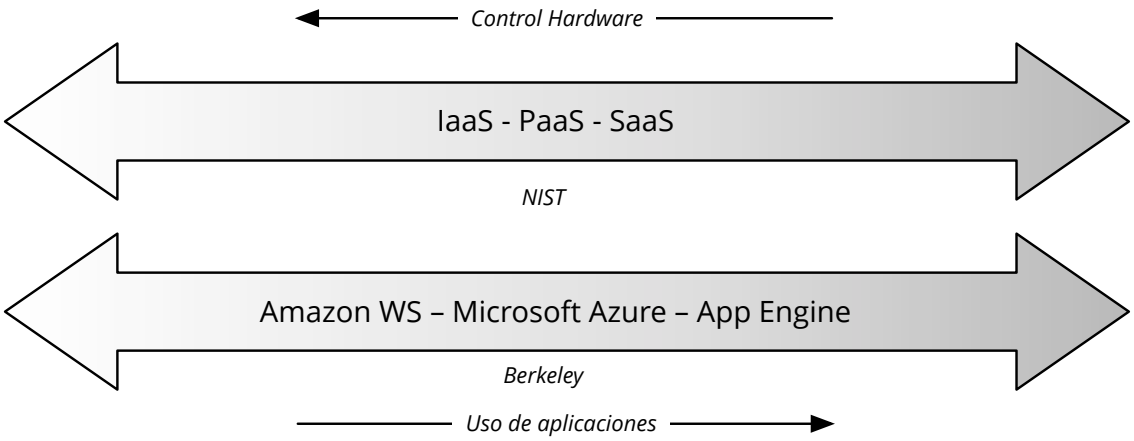


Figura 13. Comparación de la clasificaciones de servicios cloud NIST - Berkeley

5.3 Otros servicios cloud

Sobre los tipos de servicios ofrecidos en cloud computing también se ha debatido en profundidad y existen muchos términos usados a este respecto. Se habla de HaaS (*Hardware as a Service*), PaaS (*Platform as a Service*), DaaS *([Development, Database, Desktop] as a Service*), IaaS (*Infrastructure as a Service*), BaaS (*Business as a Service*), TaaS (*Testing as a Service*), FaaS (*Framework as a Service*), e incluso un genérico XaaS (*Everything as a Service*). Las definiciones asociadas a dichos términos también son muy variadas y en algunas ocasiones mal interpretadas [37]. En definitiva, muchos productos se ofertan bajo el paraguas de cloud computing y existen autores que

consideran otras divisiones y o modelos de servicio [34], [44]. Estas son normalmente especializaciones de servicio dentro de nuestra clasificación principal y a continuación explicamos algunas de ellas.

Data as a Service **(DaaS):** Los usuarios acceden a la información y datos a través de Internet. Los usuarios pueden manipular esta información remota como si lo hicieran localmente en sus ordenadores, o bien a través de una interfaz en Internet. Un ejemplo claro de este servicio lo encontramos en Google Drive [54], que permite almacenar documentos de manera remota e incluso editarlos. Otros proveedores como Dropbox [55], entre otras funciones, sincronizan una carpeta remota alojada en Internet con una local del usuario, haciendo transparente la gestión de la información al usuario. Amazon Simple Storage Service (Amazon S3) [56] proporciona una sencilla interfaz de servicios web que puede utilizarse para almacenar y recuperar la cantidad de datos que desee, cuando desee, y desde cualquier parte de la web.

Existen otros autores que refinan más en la división y consideran otro nivel más en los servicios de almacenamiento: ***Database as a Service*** [44]. En este caso, el almacenamiento se comercializa como bases de datos. Este tipo de servicios puede ser especialmente útil cuando la arquitectura de la aplicación es multi-tenant, donde se almacena información de diferentes clientes en la misma tabla. Tal y como veremos en este trabajo, será clave para el diseño de una aplicación multi-tenant la elección de la arquitectura para la capa de datos [57]. Proveedores como Amazon SimpleDB [58], Google BigTable [59] o Apache HBASE [60] ofrecen este tipo de servicio incluso con capas de abstracción y lenguajes que facilitan el trabajo de los desarrolladores.

Algunos autores engloban HaaS y DaaS [34], [37] dentro del mismo paquete y consideran solamente el concepto de Infraestructura como Servicio (IaaS).

Monitoring as a Service **(MaaS)** [44], los proveedores SaaS necesitan tener datos estadísticos fiables del uso de la aplicación. Existen proveedores que se encargan de monitorizar diferentes métricas de rendimiento de las aplicaciones u otras medidas asociadas a la satisfacción del cliente, como los Acuerdos de Nivel de Servicio (SLA: *Service Level Agreements*). En MaaS un tercer proveedor (e.g., Red Hat Command Center), analiza las aplicaciones en la red de una empresa con respecto a los SLAs y

proporciona informes de uso y rendimiento a los usuarios Otro proveedor de este tipo de servicio es Datadog [61].

Cloud Management as a Service **(CMaaS)** [44], incluye MaaS pero también añade respuestas proactivas a los eventos y no sólo se centra en la generación de informes. Cloudnexa, empresa partner de Amazon ofrece CMaaS dentro de su cartera de productos [62].

People as a Service: Consiste en la comercialización de las habilidades de programación o consultoría IT de profesionales PaaS o SaaS a través de la nube. Por ejemplo, SalesForce.com ofrece esta oportunidad a través de su portal AppExchange [63].

6 Factores importantes en cloud computing

Aunque ya hemos visto en *Orígenes del Cloud Computing* que la aparición de data centers a gran escala es la causa principal que posibilita el sueño de este paradigma, existen otras factores que también han supuesto un gran apoyo para el desarrollo de cloud computing. Los examinamos a continuación.

6.1 Virtualización

Las tecnologías de virtualización particionan el hardware y permiten ofrecer **plataformas flexibles y escalables**. Las máquinas virtuales como *VMware* [64] o *Xen* [65] ofrecen infraestructuras virtualizadas IT bajo demanda. Los avances en redes virtuales como VPN permiten a los usuarios acceder a los recursos en la nube en entornos de red personalizados.

Según [3] y en referencia a la confusión creada en torno a la definición y el entendimiento de cloud computing:

The reason why the term was not fully understood was because it was still a concept and the underlying ***virtualization*** *technology needed and our own networking infrastructure was still in its infancy which meant we could not unlock the full potential of cloud computing, until now.*

Dicha afirmación también es compartida en [35] donde se indica que cloud computing no hubiera sido posible si no es gracias al uso de la virtualización como tecnología subyacente [3]. En este mismo artículo se citan tres tipos de virtualización (ver Figura 14 [3]):

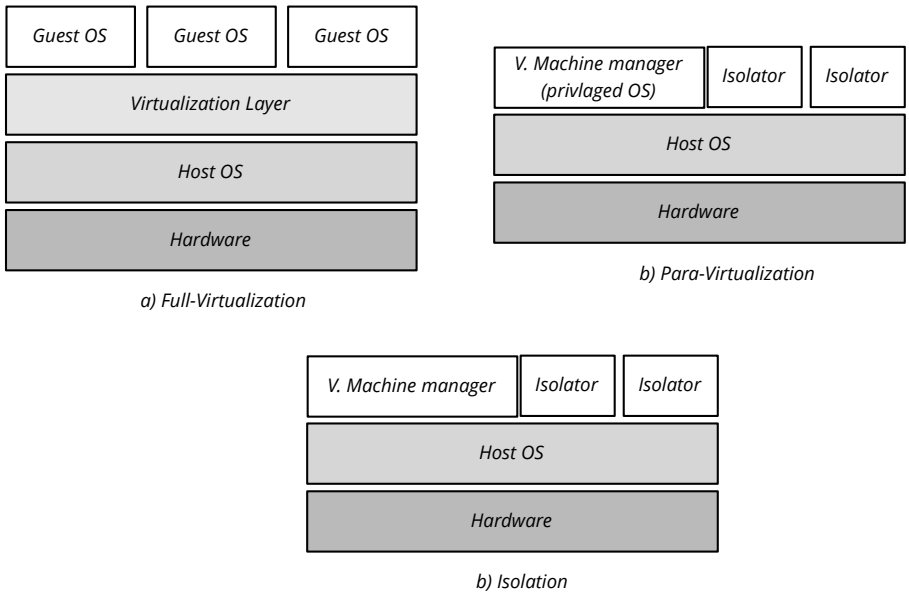


Figura 14. Tipos de virtualización

- **Full virtualization:** Este tipo de virtualización opera a nivel de procesador, el cual da soporte a diferentes sistemas operativos que son los que simulan el hardware y software para las máquinas host.

- **Para-virtualization:** Hace uso de un monitor de máquinas (*virtual machine monitor*), que es un software que permite que una sola máquina física aloje a múltiples máquinas virtuales. Por tanto, cada instancia de un programa se ejecuta de forma independiente en su propia máquina virtual.

- **Isolation**: Es similar a la para-virtualización pero que únicamente permite virtualización del mismo sistema operativo como host. Además, sólo admite Linux como sistema operativo por ser considerado el que ofrece un mejor rendimiento.

6.2 Diseño arquitectónico de los datacenters

Un centro de datos (*datacenter*) es la ubicación donde reside la fuerza de computación y el almacenamiento. Es esencial para el cloud computing y está formado por miles de dispositivos como son servidores, switches y routers. Una planificación adecuada de su arquitectura de red es crítica e influirá fuertemente en el rendimiento de las aplicaciones en este tipo de entorno distribuido. Es más, factores como la escalabilidad y recuperación a fallos deben de ser altamente considerados. En la actualidad se sigue un modelo basado en capas, que ha sido testeado en algunos de los mayores datacenters [1].

6.3 Multi-tenancy

Las arquitecturas multi-tenant (AMTs) posibilitan que varios clientes puedan compartir la misma instancia de una aplicación SaaS. De este modo, los clientes pasan a ser arrendatarios de la misma (*tenants*). Un tenant es una organización que paga por el uso de dicha aplicación, en un determinado régimen (mensual, anual, etc.) [9], [14], [16], [66]. Las AMTs son consideradas como uno de los elementos clave para el éxito de SaaS como fórmula de distribución [12], [33]. Chong [23] considera que tres atributos deben existir en una buena aplicación SaaS: escalabilidad, configurabilidad y eficiencia multi-tenant.

Multi-tenancy es pilar fundamental en esta tesis y será estudiado con más detalle en capítulos posteriores.

6.4 Web Services y SOA

Los servicios cloud son normalmente expuestos como servicios Web, los cuales siguen estándares como SOAP [67] o WSDL [68] para su publicación. Un conjunto de servicios cloud puede ser, por tanto, usado en una aplicación SOA, haciéndolos disponibles en diferentes plataformas a lo largo de Internet.

6.5 Nuevos modelos de negocio. Aumento de la confianza en Internet.

Acompañando el cambio de paradigma web, surgieron nuevos modelos de negocio que favorecen el uso de cloud computing. Estos nuevos modelos son posibles gracias a un

aumento de la confianza en Internet. Pasamos del "high-touch, high-margin, high-commitment" en la Web 1.0 al "low-touch, low-margin, low-commitment" [4] de la Web 2.0. Por ejemplo, en la Web 1.0, aceptar pagos mediante tarjeta de crédito a extraños exigía a los proveedores la firma de contratos con sistemas de pago como VeriSign o Authorize.net. Estos contratos debían de ser de largo plazo y, por tanto, poco viables para las pequeñas y medianas empresas.

Con la aparición de PayPal, cualquier web puede aceptar pagos sin una necesidad contractual fuerte y el proceso es prácticamente instantáneo. A cambio, la empresa cobra unos pequeños porcentajes de la transacción que no son diferentes a los que cobran otras empresas en transacciones habituales (Visa, por ejemplo).

Otro aspecto importante es la posibilidad de realizar **donaciones** por el uso de servicios gratuitos para aquellas personas agradecidas con el producto. Un ejemplo es la Wikipedia, cuyo mantenimiento no sería posible sin las donaciones a *Wikimedia Foundation* [69].

De forma similar, las páginas Web pueden hacer uso de Google AdSense [70] para la obtención de ingresos a través de la publicidad. De esta forma, ya no tenemos la necesidad de establecer ninguna relación con una gran compañía de publicidad como DoubleClick [71] (también adquirida por Google).

Amazon Web Services se capitalizó gracias a esta idea en el año 2006 suministrando servicios de computación bajo demanda sin ningún tipo de contrato; todo lo que los usuarios necesitaban era una simple tarjeta de crédito.

En resumen, ha habido una disminución de los compromisos en la adquisición de servicios, por no decir que en muchos casos, estos han desaparecido. Esta reducción contractual ha posibilitado el lanzamiento de nuevos proyectos que no hubieran podido ver la luz bajo el sistema anterior.

7 Obstáculos y oportunidades en cloud computing.

El crecimiento del cloud computing trae consigo una serie de dificultades. Dichos obstáculos y las oportunidades que representan para solventarlos se estudian en el

artículo "*Above The Clouds*" [4]. Veamos a continuación un resumen de estos escollos junto con sus posibles soluciones.

7.1 Disponibilidad del servicio

Obstáculo

Como cualquier otro servicio, la disponibilidad de computación es un problema si este es intermitente. Podemos pensar por ejemplo, que en nuestra empresa el servicio de teléfono no fuera continuo, sino con innumerables cortes.

Solución

Disponer de más de un proveedor de servicios. De este modo, el corte de uno puede suplirse con otro.

7.2 Datos secuestrados

Obstáculo

Los datos de los clientes se encuentran en las plataformas de los proveedores cloud. Aunque existen mejoras de la interoperabilidad de plataformas, las APIs de cada proveedor suelen ser privadas y es muy difícil que los clientes consigan trasladar la información de un proveedor a otro de manera transparente. De hecho, esta es una de las principales causas que hace que muchas compañías todavía no hayan migrado al cloud computing. Este "secuestro de datos" hace que los clientes se enfrenten a subidas de precio abusivas, problemas de fiabilidad de la empresa proveedora, o incluso el cierre del proveedor cloud, dejando huérfanos los datos del cliente.

Solución

La solución obvia es la estandarización de APIs para que el desarrollador de SaaS pueda desplegar servicios y datos entre múltiples proveedores de nube. Así, la caída de una empresa no conllevaría los datos de los clientes. El temor aquí es una reducción de la calidad buscando reducción de costes, por lo que se debe basar la elección en relación con:

- Calidad del servicio

- Combinación de nube privada y la pública, donde esta última entra en operación cuando la privada no es suficiente.

7.3 Confidencialidad y privacidad de los datos

Obstáculo

"Los datos confidenciales de mi empresa jamás estarán en la nube". Cuántas veces hemos podido escuchar esta frase de los CEO de las compañías. La oferta actual de cloud computing es pública más que privada, siendo esta más vulnerable a los ataques y provocando el rechazo a la migración por parte de las empresas. Aparte de esto, podemos encontrarnos determinadas regulaciones locales que impidan que esta información delicada se encuentre disponible en la nube.

Solución

Podemos hacer la nube igual de segura que otros entornos informáticos como por ejemplo las redes locales (LAN). El mayor problema radica en la confianza de los usuarios y los Gobiernos en la seguridad de la nube. Una posible solución es la encriptación de datos. El problema se ve acrecentado cuando el software SaaS es multi-tenant, esto es, cuando los datos de los clientes comparten la misma base de datos.

Analizamos también las normas que obligan a los proveedores SaaS a mantener la información de los clientes dentro de los límites territoriales del país. Los proveedores cloud dan libertad a sus clientes acerca de dónde almacenar sus datos. Por ejemplo, Amazon da a escoger a sus clientes entre alojar la información en América o en Europa, al disponer de centros de datos en ambos continentes. Veamos el contenido de la web que publicita los servicios S3 [56]:

Un depósito puede estar almacenado en una de varias Regiones. Debe escoger una Región cercana para optimizar la latencia, minimizar los costes o afrontar exigencias reguladoras. Amazon S3 está disponible en las regiones EE.UU. Estándar, EE.UU. Oeste (Oregón), EE.UU. Oeste (Norte de California), UE (Irlanda), Asia Pacífico (Singapur), Asia Pacífico (Tokio), Asia Pacífico (Sídney), América del Sur (São Paulo) y GovCloud (EE.UU.). La Región EE.UU. Estándar redirige automáticamente las

solicitudes hacia instalaciones situadas en el norte de Virginia o en el noroeste del Pacífico por medio de asignaciones de red.

7.4 Cuellos de botella de la transferencia de datos

Obstáculo

Las aplicaciones consumen cada vez una mayor cantidad de datos y la cantidad de información que se transfiere es cada vez mayor. Un coste medio de 100€ por Terabyte (Amazon) transferido no es viable y es necesario buscar otras alternativas.

Solución

Jim Gray encontró que la solución más sencilla a este cuello de botella es el envío de la información por la vía tradicional de mensajería [72]. Aunque no se pueda asegurar la fiabilidad en la entrega por parte de las empresas de transportes, Gray experimentó un sólo fallo en 400 envíos.

Analicemos con números esta opción. Supongamos que necesitamos enviar 10TB de datos. Suponiendo una velocidad de 20 Mbit/sg. sobre una red WAN tardaríamos: 10 × 1012 Bytes / (20×106 bits/segundo) = (8×1013)/(2×107)= 4,000,000 segundos; más de 45 días. Amazon cobraría además una media de 1000€ por este proceso. Si, por el contrario, enviamos 10 discos de 1TB por transporte, tardaríamos un día (de media para un traslado nacional) y apenas nos costaría 300€.

Otra solución diferente consiste en mantener la información en la nube. Por una vez, la información la tenemos en la nube y no tenemos por qué moverla de ahí. Los proveedores cloud, pueden encontrar otras formas de obtener beneficios, diferentes al consumo de ancho de banda. Por ejemplo, Amazon mantiene gratis la base de datos del censo, producto que puede favorecer el consumo de ciclos de CPU para aquellos que quieran utilizarla.

Como tercera solución está la reducción drástica del precio del ancho de banda. El principal coste de las transmisiones (dos tercios) viene dado por los routers “high end” mientras que el tercio restante es debido al cableado. Existen ya investigaciones avanzadas que pretenden reducir el coste de los routers buscando alternativas de bajo coste. Si una vez encontradas estas vías alternativas, los proveedores procedieran a un

cambio en sus infraestructuras, veríamos reducido el coste de las transmisiones de manera considerable.

7.5 Rendimiento imprevisible

Obstáculo

Las máquinas virtuales comparten bien CPUs y memoria, sin embargo, la distribución del E/S (entrada/salida) es más problemática. Mientras que la escritura en memoria tiene un ancho de banda promedio de 1355 MB/s, la escritura en disco tiene un ancho de banda de 55 MB/s. Por tanto, tenemos un obstáculo de compartición de E/S entre máquinas virtuales que hace imprevisible el rendimiento de las mismas.

Solución

Una solución es la mejora de las arquitecturas y sistemas operativos para aumentar el rendimiento de trabajo compartido en disco. Tecnologías como PCI Express son difíciles de virtualizar, pero sin embardo de especial importancia para el cloud computing. Los mainframes de IBM resolvieron este problema en los 80 por lo que se puede aprender de esas arquitecturas.

Otra oportunidad es la mejora de las memorias flash, para que disminuyan las interferencias de E/S. Las memorias flash pueden soportar muchas más interrupciones de E/S que los discos convencionales.

Otro obstáculo relacionado con el rendimiento es el uso de la nube para el procesamiento de trabajos con alto requerimiento de computación (HPC, High Performance Computing). El problema radica en que las aplicaciones HPC precisan que todas sus hebras se ejecuten de manera simultánea, cosa que hoy en día no pueden asegurar las máquinas virtuales del cloud computing. La oportunidad aquí, consiste en ofrecer algo parecido al "*Gang Scheduling*" [73] para el cloud computing. Se trata de algoritmos de planificación para aplicaciones paralelas que permiten a los procesos o hebras ejecutarse simultáneamente en diferentes procesadores.

7.6 Escalabilidad de almacenamiento

Obstáculo

Los principales factores que hacen atractivo el cloud computing son: elasticidad (scale-up o down dependiendo de las necesidades en ese momento), no tener costes preestablecidos (pago por uso) e infinita capacidad bajo demanda. Mientras que es sencillo entenderlos al aplicarlos a la capacidad de computación es complicado aplicarlo a la capacidad de almacenamiento persistente.

Solución

Aun no se ha encontrado solución a este problema. Tenemos por tanto la oportunidad de conseguir sistemas de almacenamiento que se adapten a estas necesidades.

7.7 Fallos en sistemas distribuidos de gran escala

Obstáculo

La reproducción de fallos para sistemas distribuidos a gran escala no puede ser realizada en sistemas de configuración menor, por la propia naturaleza de los mismos. Por tanto, el testeo ha de ser realizado en plataformas de escala semejante, con el aumento de costes económicos y de complejidad que ello supone.

Solución

La recreación de esos ambientes distribuidos se puede llevar a cabo mediante máquinas virtuales. Muchos proveedores de SaaS no estaban usándolas, pues no confiaban en su rendimiento. Dado que las MVs ahora son obligatorias en el utility computing, será posible por tanto crear estos ambientes.

7.8 Rapidez en la escalabilidad

Obstáculo

El pago por uso (pay-as-you-go) se aplica al ancho de banda y al almacenamiento contando el número de bytes consumidos. En el caso de la computación, el recuento es diferente, dependiendo del nivel de virtualización. Google AppEngine responde escalando arriba o abajo por aumentos o disminución de la carga. Los usuarios pagan por el número de ciclos consumidos. AWS, sin embargo, cobra por el número de instancias por hora, incluso si la instancia está sin usar (*idle*). Un ordenador ocioso

consume dos tercios de lo que consume uno ocupado, por lo tanto, debemos buscar la optimización en el escalado no sólo para la conservación de recursos, sino para la reducción del coste de servicio. Además, esta búsqueda por la optimización de los recursos, obligará a los programadores a ser lo más minuciosos posibles para la maximización de uso y disminución de costes, dando como resultado aplicaciones perfectamente optimizadas.

Solución

Lograr la escalabilidad automática en respuesta a la carga requerida a fin de ahorrar recursos y dinero. Además, estaremos consiguiendo una mejora indirecta en la calidad de programación de aplicaciones.

7.9 Diferentes usuarios, misma reputación

Obstáculo

En cloud computing un mismo recurso es usado por diferentes usuarios consumidores y no existe propiedad ni exclusividad en los mismos. Por ello, el mal comportamiento de uno de los consumidores afecta el resto. Por ejemplo, poner en la lista negra una determinada IP de un EC2 debido a malas prácticas (Spam, por ejemplo), limita el tipo de aplicaciones que podemos tener alojadas en la máquina para el total de los usuarios. Podemos decir que la **reputación no virtualiza bien en la nube**.

Solución

Creación de servicios de protección de reputación. Otro aspecto a tener en cuenta es la posible migración de responsabilidad a la compañía consumidora del servicio y no al proveedor cloud.

7.10 Licencias Software

Obstáculo

La tipología de las licencias actuales de software limita en la mayoría de los casos el tipo de ordenadores en los que las aplicaciones pueden correr. Los consumidores pagan por el software y luego un mantenimiento anual. Los proveedores SAP (*Software*

Applications Providers) han anunciado que aumentarán el coste de adquisición del software en un 22%, lo cual equipara los precios de Oracle. Esta subida provocará, por tanto, que muchos proveedores Cloud se pasen al uso de Software Libre debido a que el uso de versiones comerciales de software no son compatibles con Utility Computing.

Solución

Una primera solución es mantenerse del lado del software libre, o bien conseguir que las empresas distribuidoras de software rebajen sus expectativas de ganancia, adoptando un modelo que se adapte mejor a cloud computing. Por ejemplo, Microsoft y Amazon ahora ofrecen una licencia pago-por-consumo para Microsoft Server y Windows SQL en las máquinas EC2. Una instancia de EC2 con este software cuesta 0,15$, frente a los 0,10$ de aquellas con licencia de software libre.

Asociado a este problema tenemos el de convencer a las grandes empresas de software de esta nueva fórmula de comercialización en régimen de pago-por-uso (pay-as-yo-go).

Capítulo II

Software as a Service (SaaS)

Este capítulo tiene como objetivo exponer de forma clara el concepto de SaaS, incluyendo sus orígenes, definición y tipos. Se describirá su diferencia con el software tradicional, así como un conjunto de aspectos importantes a tener en cuenta. Finalmente, se discuten las ventajas e inconvenientes de este nuevo modelo de comercialización de software.

1 Introducción

Las aplicaciones de escritorio y las de Internet están sufriendo un cambio importante: se están mezclando [74]. La velocidad de acceso, generalización de uso de Internet, y los avances en la tecnología Web (HTML5, CSS3, jQuery, jSON, XML, etc.), han permitido que las aplicaciones web puedan ser tan interactivas, usables y bellas [75] como las de escritorio.

SaaS establece **un nuevo modelo para construir un negocio en base a estos avances**. En SaaS, las aplicaciones son accedidas a través de Internet, a un coste más reducido. En contraposición, dejamos de ser dueños de las aplicaciones. En SaaS, las aplicaciones son un servicio y nosotros, sus suscriptores.

Si nos paramos a pensar, son muchos los programas que usamos diariamente de forma exclusiva por Internet; clientes de correo, editores de texto, redes sociales etc. Hoy en día, las grandes empresas de software presentan la misma aplicación del programa tanto en versión escritorio como versión web (Spotify, Facebook, Twitter, Google Apps,…). Aún así, las versiones de escritorio precisan de computación en la nube, esto es, sin conexión a internet nunca seremos capaces ejecutarlas correctamente. Las versiones locales necesitan como mínimo acceder a nuestros datos; la interfaz, la capa de presentación es la que normalmente se instala localmente en nuestros ordenadores. No tener la versión de escritorio no es ya importante, no sólo porque a través de una web podemos acceder a ellas, sino porque también la descarga e instalación de las versiones de escritorio las podemos hacer en cuestión de minutos. Las aplicaciones ya no son pesadas como antes (decenas de discos, CDs o DVDs), sino que tan sólo es necesario bajarse unos poco MBs a través de la página del proveedor o desde repositorios especializados.

A nivel empresarial, el modelo es atractivo. En vez de pagar cantidades ingentes de dinero por la compra de un *Enterprise Resource Planning* (ERP) o *un Customer Relationship Manager* (CRM) e instalarlo localmente en nuestros servidores, nos suscribimos a un servicio que ofrece un proveedor, el cual se encarga del mantenimiento y de la infraestructura TIC, a cambio de un precio asequible basado en cuotas. Desde el punto de vista del proveedor el mercado se amplia, no sólo por la

disminución de precios sino por las nuevas posibilidades de alcance que nos brida Internet.

SaaS es un nuevo modelo de distribución de software que supone ventajas tanto para los individuales, como para las empresas cliente y proveedores de software.

2 Orígenes

2.1 The Pennine Group

En el año 1999, el "*Pennine Group*", un consorcio de Ingenieros e investigadores de la *Universidad de Durham, Keel y UMIST* predijeron que el futuro del software sería un mercado en el que los servicios son provistos según demanda a los clientes. Esta idea fue acuñada con el término *Software as a Service* (SaaS, Software como Servicio) [76].

En el año 2003, [77] propuso que la esencia el modelo SaaS estaba en **separar los conceptos de posesión del software y el de su uso**. La funcionalidad de un software se puede definir como un conjunto de servicios que se configuran y enlazan en el momento de la entrega. Se confirma un cambio de responsabilidad en el "*know-how*" del negocio, que ya no depende de saber qué proveedor (aunque también inevitable), sino de saber qué servicios se necesitan en un determinado momento y negociar unos términos adecuados para su uso. Con las nuevas tecnologías, problemas existentes en el software tradicional como el despliegue y la evolución podrán ser resueltos.

En Junio del año 2006, gracias al soporte de la *Web 2.0*, Salesforce [53] lanzó su CRM bajo demanda basándose en la idea de SaaS y en la teoría del *Long Tail* de Chris Anderson [78]. Al mismo tiempo, compañías de software tradicional también empezaron a adentrarse en el negocio SaaS y comenzaron a explotarlo. En Noviembre de 2006, Microsoft presentó un software llamado *Live online service*. En Julio de 2007, Oracle anunció el lanzamiento de la última versión de *Siebel hosted CRM* en el mercado chino. En Septiembre de 2007 SAP, el mayor desarrollador de ERP, presentó el software SaaS A1S.

2.2 Del ASP al SaaS

En la década de los 90, junto con el desarrollo de Internet, apareció el concepto *Application Service Provider* (ASP), por el que una empresa podía acceder remotamente a aplicaciones gestionadas por un tercero, el proveedor ASP.

Dichas aplicaciones se basaban en una plataforma cliente-servidor y tan solo hacían uso de algunas páginas HTML estáticas que no estaban diseñadas para su uso web. El servicio se implementaba de forma que cada cliente accedía a su propia base de datos dedicada (con su propia licencia) por lo que la administración de la infraestructura era compleja y las actualizaciones se debían coordinar adecuadamente con todos los clientes.

Por ello, este modelo de negocio **no se desarrolló plenamente**. Aunque era una estrategia válida, aún no se disponía de la tecnología adecuada para optimizar los costes y ofrecer un buen servicio. A primera vista parece no haber diferencia entre ASP y SaaS, pero sin embargo si las hay y son las que han determinado el destino de cada una de ellas.

El modelo ASP se basaba en externalizar determinadas funciones de procesamiento de una empresa en terceros, sin tener en cuenta la posibilidad de compartir ese mismo servicio con otros clientes (multi-tenancy) [74] o de la personalización de la aplicación [79].

Además de esto, había poca profesionalidad de las ASP en el conocimiento profundo del negocio, teniendo en cuenta la importancia del software para los procesos de la empresa adquisitoria.

En el modelo SaaS, las aplicaciones han sido diseñadas específicamente para el acceso web, teniendo en cuenta aspectos primordiales como la experiencia de usuario. ASP en cambio, presentaba inmensas aplicaciones cliente-servidor con interfaces web simples de escasa usabilidad. Por regla general, las aplicaciones ASP dieron un mal resultado, debido a las prisas de los vendedores en tener en producto en el mercado, sin optimizar el rendimiento, seguridad y configurabilidad.

Según Christopher Souza, director de New England Data Services (NEDS) [80]: *"Con SaaS, estos problemas se han solucionado. Este hecho, unido a la disminución de costes y al cambio de conciencia de los usuarios, hace que sea un momento perfecto para las empresas en adoptar el modelo SaaS"*.

Según [74], las ASP fracasaron en la Web 1.0 por dos razones. En primer lugar porque no cambiaron la arquitectura del software (como lo hacemos ahora con el uso de nuevas tecnologías como multi-tenancy, Ajax, XML, Web 2.0 etc.). Simplemente se dedicaron a vender aplicaciones tradicionales a empresas que no las querían tenerlas alojadas en sus propios sistemas. El coste de alojamiento fue algo que los proveedores ASP no pudieron soportar. En segundo lugar, porque sólo un pequeño segmento del mercado quería realizar la externalización de esos servicios. Consideraba que las aplicaciones de negocio eran un activo estratégico y preferían tenerlos instalados de forma local.

De acuerdo a un estudio de Gartner [81], está previsto que SaaS se convierta progresivamente importante en la mayor parte del software empresarial. Algunos proveedores TIC, han vaticinado el software-as-a-service como un excelente complemento al software-on-premises ya que soluciona los problemas que previamente habían presentado otras soluciones bajo demanda como ASP.

3 Definición

Software as a Service (SaaS) es uno de los modelos de servicio existentes cloud computing.

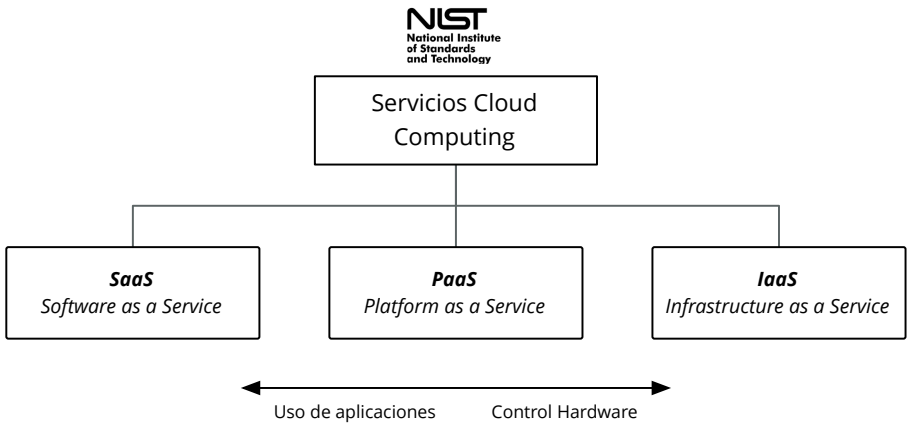


Figura 15. Principales servicios cloud según NIST.

La Figura 15 recuerda la clasificación de servicios cloud según NIST, en la que se define SaaS como [33]: *"The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user- specific application configuration settings"*.

En resumen y según esto, podemos definir SaaS como un nivel de servicio en cloud computing en el que la prestación suministrada a los usuarios es el uso de aplicaciones de un proveedor, que son ejecutadas en una infraestructura cloud. A las aplicaciones se puede acceder bien a través de un clientes ligero (*thin clients*), como un navegador web (e.g, email basado en web) o a través de un programa. El cliente no gestiona ni controla la infraestructura cloud subyacente incluyendo redes, servidores, sistemas operativos o almacenamiento e incluso las mismas aplicaciones, con las excepción de un limitado número de opciones de configuración a nivel de usuario final.

A pesar de que algunos autores [23], [82] consideran que no existe una definición general del concepto, a día de hoy la definición NIST es la más extendida. Además, a diferencia del caso de cloud computing, las enunciaciones que podemos encontrar en diferentes trabajos son bastantes similares. Veamos ejemplos de **otras definiciones existentes:**

[23] define SaaS de una manera escueta: *"Software deployed as a hosted service and accessed over the Internet."*. Esto es : "software desplegado como un servicio alojado al cual se accede a través de Internet". Esta definición no describe un tipo de arquitectura específica, ni tampoco cita tecnologías y protocolos a usar. Tampoco diferencia entre servicios orientados al negocio, o servicios orientados al cliente, etc.

De acuerdo con esta definición, la clave que caracteriza al SaaS se encuentra en dónde reside el código y cómo es implementado y accedido

Del mismo modo, también existe una definición concisa en [83]: "*SaaS consiste en un acceso online independiente del tiempo y localización a una aplicación gestionada de forma remota"*

En [43] SaaS se define como un modelo de distribución del software que proporciona a los clientes el acceso a aplicaciones a través de la Internet. El software se suministra como un servicio, de manera que el usuario no tiene que preocuparse del mantenimiento de dichas aplicaciones. Para el usuario, este modelo permite optimizar costes y recursos. Para el suministrador de software, este modelo permite implementar economías de escala optimizando los costes.

En [84] podemos leer: *"SaaS es un modelo de distribución de software que facilita acceso remoto a los clientes de funcionalidades de negocio (normalmente vía Internet), proporcionando este acceso como un servicio"*. A pesar de estar en una definición dentro de la línea, en este caso los autores se separan en dos aspectos. Primero consideran que las funcionalidades accedidas son de negocio (empresariales) y segundo que el acceso se realiza "normalmente" a través de Internet.

Los autores de [82] configuran un compendio de ocho definiciones de SaaS, llegando a la conclusión de que son **cinco las características comunes** que presenta este modelo:

1. El producto se accede a través de un navegador web
2. El producto no esta hecho a medida para cada cliente (a diferencia de ASP), luego veremos como se puede conseguir personalización de la aplicación gracias al uso de *metadatos*.
3. El producto no incluye software adicional que necesite ser instalados localmente en los ordenadores del cliente.
4. El producto no requiere especial trabajo de instalación e integración.
5. El precio del producto depende del grado de uso del sistema.

En resumen, aunque existen muchas definiciones de SaaS por parte de distintos investigadores, todas ellas son semejantes aunque con pequeñas distinciones. Hoy en día se toma como referencia y consenso la primera definición (NIST) citada en esta sección.

4 Clases de SaaS

Por su definición SaaS incluye una serie de aplicaciones que quizás no nos esperábamos encontrar en esta categoría. Por ejemplo, consideremos servicios mail, como *Gmail* o *Hotmail (ahora Outlook)*. Aunque quizás no sean el primer ejemplo que se nos venga a la cabeza cuando pensemos en SaaS, cumplen con todos los criterios. Por un lado existe un proveedor que suministra el servicio y que mantiene la lógica del programa y los datos. Por otro, el sistema es accedido por los usuarios finales a través de Internet, mediante un navegador. Siguiendo este criterio, **dependiendo del modelo de negocio** podemos categorizar las aplicaciones SaaS en dos [7], [23]:

- **Orientadas al negocio (*Line-of-business services, LOB*)**: Se ofrecen a organizaciones y empresas de todos los tamaños. A menudo, este tipo de servicios son grandes soluciones empresariales configurables, destinadas a facilitar procesos de negocio como la contabilidad y finanzas, relaciones con el cliente, gestión de pedidos, etc. Las aplicaciones orientadas al negocio normalmente se comercializan en régimen de suscripción pagando una determinada cuota.

- **Orientadas al usuario (*Customer-oriented-services*):** Ofrecidas al público en general. Al igual que las soluciones empresariales se pueden comercializar por suscripción, aunque también es muy frecuente que sean de coste cero, estando financiadas por la publicidad y anuncios que soportan.

Conceptos importantes como multi-tenancy, extensibilidad y escalabilidad, íntimamente relacionados con SaaS y que estudiaremos más adelante pueden ser aplicados a ambos tipos de aplicaciones.

Es difícil que las grandes empresas encuentren un software SaaS que complete todas sus necesidades funcionales. Estas disponen de sistemas complejos, realizados a medida y tan específicos que no encajarían dentro de un modelo multi-tenant compartido. Según esto, podemos establecer una división del SaaS **en función del tipo de servicio ofrecido** [85]:

- **Plataformas de servicios:** En este caso no se ofrecen funcionalidades completas. Las empresas suministran servicios de granularidad menor que son utilizados como

herramientas y que se integran de forma transparente en otros sistemas (vía APIs de web services: SOAP, JSON, REST, etc..). Estas aplicaciones son desarrolladas por empresas terceras (o por la misma) que son las que comercializan la solución software y usan la plataforma para mejorar sus soluciones empresariales. *Google API* [86] es un ejemplo de este tipo; ofrece por ejemplo consultas a su base de datos de Google Maps y su precio depende del número de consultas diarias. Las 25.000 primeras consultas son gratuitas.

- **Servicios completos**: Las empresas proveedoras ofertan aplicaciones completas que son desarrolladas por sus propios equipos de desarrollo y sus clientes son el usuario final. Dichas aplicaciones también pueden proporcionar APIs de servicios web. Ambas tipologías no son excluyentes pero normalmente estos accesos SOA se usan no para crear nuevas soluciones empresariales, sino para soluciones complementarias (App móvil, por ejemplo)

En [87], los sistemas de información son clasificados en función del tipo de estrategia para comercializar al target destino de la aplicación:

- Una **estrategia horizontal** es aquella en la que las empresas desarrollan un software que pueda ser de utilidad a la mayor parte de los usuarios, independientemente del sector al que pertenezcan.

- Por el contrario, en una **estrategia vertical** las aplicaciones desarrollan funcionalidades muy específicas de sector, estando altamente especializadas para ese mercado.

5 SaaS y el software tradicional

SaaS supone un cambio radical en el modelo de distribución de software frente al modelo tradicional. En esta sección analizados las principales diferencias entre ambos modelos.

5.1 Propiedad del software

Gran cantidad de software continua vendiéndose de la forma tradicional, esto es, el cliente compra una licencia de uso, instala el programa de forma local y el proveedor realiza el mantenimiento y soporte según los términos de licencia [13]. En este modelo tradicional, el concepto de posesión de software es un poco engañoso. Legalmente el cliente sólo adquiere una licencia de uso de software con la que legalmente tiene derecho a la instalación de una copia para utilizarlo. Sin embargo, en la práctica el cliente es el propietario del software, puede usarlo tantas veces y durante el tiempo que sea necesario.

En el modelo SaaS, el cliente hace uso de un software en régimen de suscripción. No es el propietario del software, sino que contrata el servicio. Si el cliente deja de pagar el servicio, entonces el software desaparece. El propietario es la empresa proveedora de software. La propiedad por tanto, supone un punto económico a favor del desarrollador, con respecto al modelo tradicional.

5.2 Single-tenant vs. Multi-tenant

Single-tenant: El modelo tradicional de software es un modelo aislado de un único cliente (*single-tenant,* aunque por su naturaleza de pago de licencia podría llamarse *single-owner* o *multi-instance*). Esto significa que un único usuario compra la aplicación y la instala en un servidor propio. Este servidor aloja y despliega la aplicación sólo para el grupo de usuarios finales del cliente. La mayor parte de las aplicaciones comercializadas hoy en día bajo esta fórmula [88] (Figura 16-a).

Multi-tenant: SaaS presenta una arquitectura software multi-tenant. Esto quiere decir que la misma aplicación es compartida por diferentes empresas [9]. Una aplicación multi-tenant permite la personalización de la aplicación para atender las necesidades de los clientes. De este modo, los proveedores son capaces de dar servicio a cientos de empresas suscriptoras; los clientes por su parte no son conscientes de esta situación ya que para ellos la multiplexación ocurre de manera transparente, al tener la impresión de disponer de una aplicación dedicada para su organización [14]. En SaaS los clientes no son dueños de las aplicaciones, sino usuarios de las mismas (Figura 16-b).

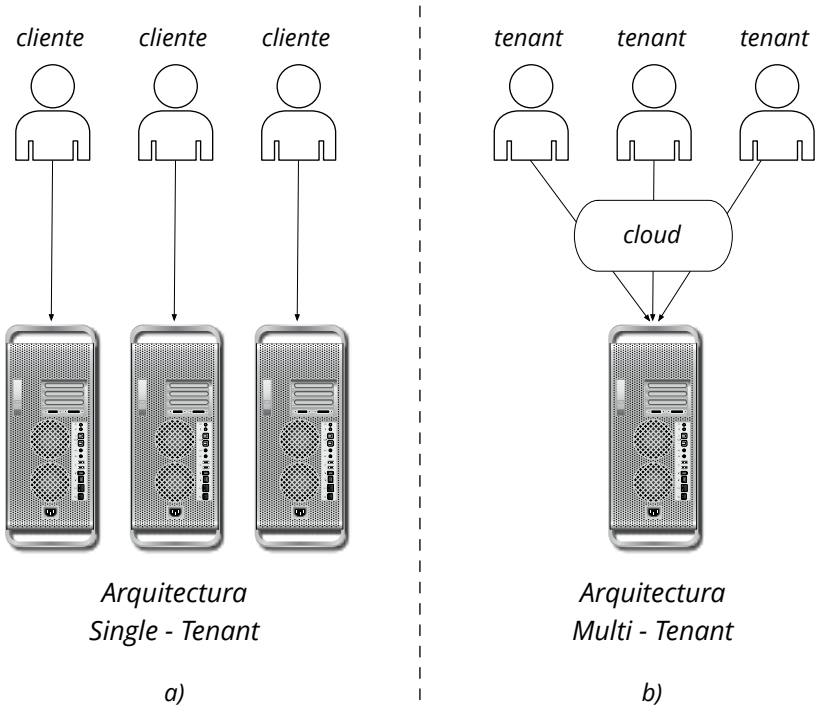


Figura 16. Comparativa de arquitecturas Single vs Multi-tenant

Las aplicaciones ASP que estudiábamos anteriormente son típicamente single-tenant con la diferencia de ser alojadas en un proveedor externo. ASP consiste en aplicaciones cliente-servidor con *frontends* HTML que permiten un acceso remoto a la aplicación [88]. Cada cliente tiene su propia implantación y mantenimiento lo cual aumenta los costes y el esfuerzo de parte del proveedor. En contra, las aplicaciones SaaS son en realidad una sola instancia que es compartida entre los diferentes tenants. Además están alojadas en el cloud pero por proveedores expertos en la materia ya que entre otros aspectos, son los dueños de la aplicación.

5.3 Software on/off premises

Según [23], todos los autores coinciden en un principio fundamental que distingue el software empaquetado tradicional del nuevo modelo: "En SaaS el *Software es implementado como un servicio alojado y accedido a través de Internet*."

Así podemos considerar que en el modelo anterior, el software se instala de forma local en nuestras instalaciones (*on-premises software*) mientras que en SaaS los sistemas son instalados en instalaciones externas (*off-premises* software). El concepto on/off

premises está ligado a los estudiados anteriormente. Las aplicaciones on-premises son instalaciones normalmente single-tenant mientras que las multi-tenant son off-premises.

En SaaS las aplicaciones se ofrecen como servicios más que como paquetes software que se compran de manera individual. La información de los clientes se encuentra almacenada también en la nube, normalmente bajo el modelo IaaS (DaaS si se quiere ser más específico). **Es obvio que desaparece el concepto de licencia de pago único y se pasa al concepto de pago por uso, mensualmente, anualmente, etc.**

Una de las claves de este modelo "externo a nuestras instalaciones" reside en la compartición del sistema por parte de los diferentes suscriptores. Este concepto que ya hemos presentado anteriormente se muestra en la Figura 17 y se conoce como **multi-tenancy.** Gracias a multi-tenancy, SaaS posibilita la reducción de costes por parte de los proveedores, que sólo necesitan mantener una aplicación (además de actualizaciones, etc.). Los clientes además de ver reducidos los costes operacionales (*OPEX, operational expenses*), apenas tienen inversiones capitales iniciales (*CAPEX, capital expenses*) al no necesitar adecuar su infraestructura TIC ni realizar inversión inicial por la compra de la aplicación.

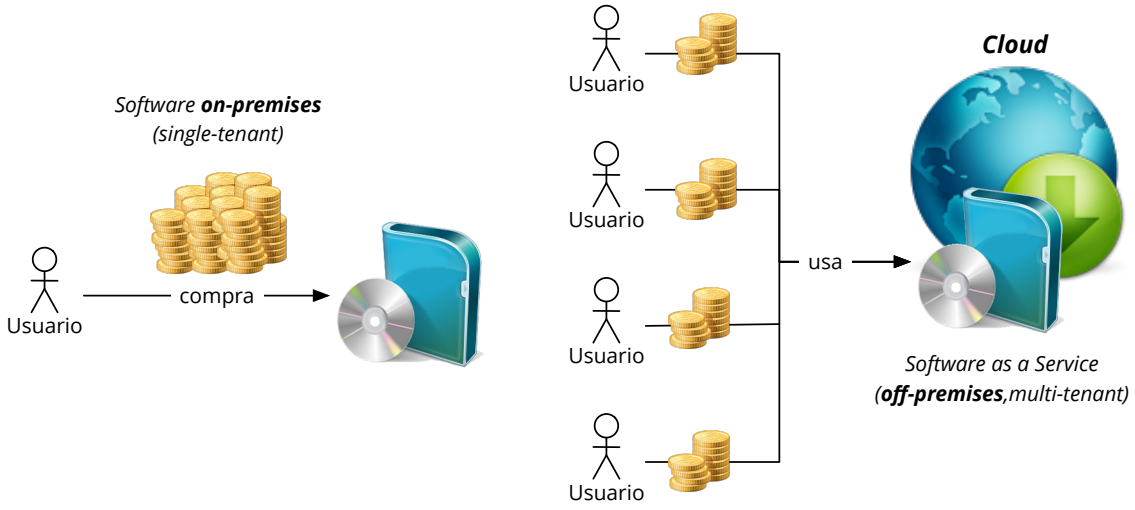


Figura 17. Software on-premises vs. SaaS (off-premises)

6 Principales aspectos en SaaS

6.1 Modelos de suscripción

Según [89], existen varios modelos de suscripción que pueden dividirse en cinco categorías:

- Modelo basado en **suscripción**. Con este modelo los clientes pagan una cuota de mensual basada en el uso del sistema. Normalmente también se tienen en cuenta el número de usuarios finales de la aplicación.

- Modelo de tasa **fija**. Este modelo es semejante al anterior. La diferencia radica en que las cuotas mensuales de pago se fijan y pagan por adelantado en función del número de usuarios, niveles de soporte y módulos de la aplicación utilizados.

- Modelo basado en **uso**. En este caso la facturación está relacionada con cuánto usamos el sistema y los picos de uso. Este método también tiene en cuenta el número de servidores utilizados así como los usuarios concurrentes.

- Modelo basado en **transacciones**. Las empresas cobran por cada transacción realizada en la aplicación.

- Modelo basado en **valores**. La cantidad de dinero que se paga al vendedor es fijada en función de una serie de objetivos conseguidos por parte del cliente.

6.2 Multi-tenancy

En la definición ya vista de SaaS: "*Software implementado como un servicio alojado que es accedido a través de Internet"* [23] nos encontramos con términos como *software* y *acceso*. Ambos son amplios y pueden incluir muchas aspectos diferentes, y tal vez demasiados. Para un arquitecto software, la definición en realidad no arroja ninguna luz de por qué funcionan las cosas, lo que diferencia una aplicación SaaS con éxito de otra que no lo tiene. Una aplicación LOB con código de más de una década de antigüedad y un frontend improvisado HTML podría encajar perfectamente en esta definición. Sin embargo, la mayoría de las aplicaciones entran en problemas cuando no

pueden escalar bien, o el coste de las mismas no es rentable. Por tanto, no todas las aplicaciones que se acceden por web, deben de entrar en el marco de SaaS.

Multi-tenancy sea quizás el mayor cambio de paradigma para un arquitecto software acostumbrado al diseño aislado de aplicaciones single-tenant (mono-usuarios). En SaaS, cuando un usuario por ejemplo se conecta a un CRM, la instancia de la aplicación a la que se conecta también es válida para decenas, incluso cientos de usuarios con las mismas necesidades, de cualquier compañía. Esto ocurre de manera transparente, sin que el usuario se de cuenta de ello.

Multi-tenancy obliga a la arquitectura de la aplicación a maximizar el uso compartido de recursos entre diferentes tenants, pero diferenciando los datos que pertenecen a cada uno de ellos. Uno de los problemas principales en el es la personalización de la aplicación. Si realizamos cambios en el código para adaptar las necesidades de un determinado tenant, los cambios serán vistos por el resto. Esto se soluciona mediante el uso de metadatos por parte de los arrendadores que permitan la **configuración individual del software**.

6.3 Seguridad y privacidad

La información es probablemente el mayor activo de las empresas. Si las empresas quieren beneficiarse de SaaS, deben estar dispuestas a trasladar el control a un proveedor tercero, en el cual deben confiar ciegamente para la protección de sus datos. Una aplicación SaaS está alojada off-premises lo cual significa para las empresas que está fuera de su control de seguridad, externo a la protección de sus firewalls. Esto representa un escollo grande en la adopción de SaaS, preocupadas en accesos indeseables a su información [89].

Según [90] las empresas proveedoras cloud debe de marcarse unos principales objetivos en seguridad entre los cuales podemos incluir:

- **Proteger los datos** de los clientes de accesos no autorizados, divulgación, monitorización o modificación.

- **Prevenir los accesos** no autorizados a infraestructura. Los proveedores deben de implementar dominios seguros con una separación lógica de los recursos informáticos. Esto implica entre otras cosas una reserva y control de la capacidad de carga de trabajo de clientes para evitar que los picos de procesamiento de unos clientes afecten al resto.

- Desplegar **aplicaciones en el cloud** que hayan sido diseñadas y programadas para un modelo de acceso vía Internet con sus correspondientes amenazas.

- Prevenir que los **navegadores web** mitiguen la seguridad de los usuarios finales. Esto incluye el uso de firewalls, software de seguridad y un correcto mantenimiento en los ordenadores personales.

- Incluir **control de acceso**, detección de intrusos y soluciones preventivas en las implementaciones.

- Definir **límites de confianza** entre los proveedores y los clientes y asegurar así que las responsabilidades para implementar controles de seguridad están claramente identificadas.

- Implementar **APIs** estandarizadas para la interoperabilidad y la portabilidad que aseguren una fácil migración de los datos del cliente a otro proveedor.

Según [90], existen dos tipos de información que los proveedores cloud deben proteger y asegurar un correcto uso, disposición y comunicación: *Información personal (IP)* e *Información de identificación personal (IIP)*; esta última se refiere a aquella información que puede ser usada para usada para trazar y descubrir las entidades de los individuos (nombre, número de la seguridad social). El *Consejo de jefes de nuevas tecnologías del Comité de privacidad de EEUU* (*CIO Council Privacy Comitte*) considera la protección de este tipo de información (IIP) como una prioridad imperativa del gobierno federal.

Por tanto, para acelerar la aceptación de cloud computing y avanzar en el desarrollo de los servicios cloud, se debe trabajar en soluciones a los problemas de seguridad que

aumenten el grado de confianza de los clientes. La seguridad y autenticación aseguran la **privacidad de los datos**, separando los que son de cada cliente y evitando accesos indeseables [23].

En el capítulo relativo a multi-tenancy estudiaremos con mayor profundidad aspectos relacionados con la seguridad y privacidad. Analizaremos este concepto a nivel de base de datos y repasaremos diferentes posibilidades de implantación.

6.4 Metadatos, personalización y configurabilidad

En SaaS, las aplicaciones son compartidas entre los diferentes clientes suscriptores. La arquitectura interna multi-tenant, permite esta multiplexación de forma que los diferentes tenants consideran que están ejecutando una aplicación dedicada para su organización. Sin embargo cada empresa puede tener unas necesidades diferentes, incluso perteneciendo al mismo sector. Las aplicaciones SaaS deben poder permitir a cada arrendatario tener una cierta personalización para conseguir que la solución se adapte a sus necesidades sin necesidad de duplicar el código.

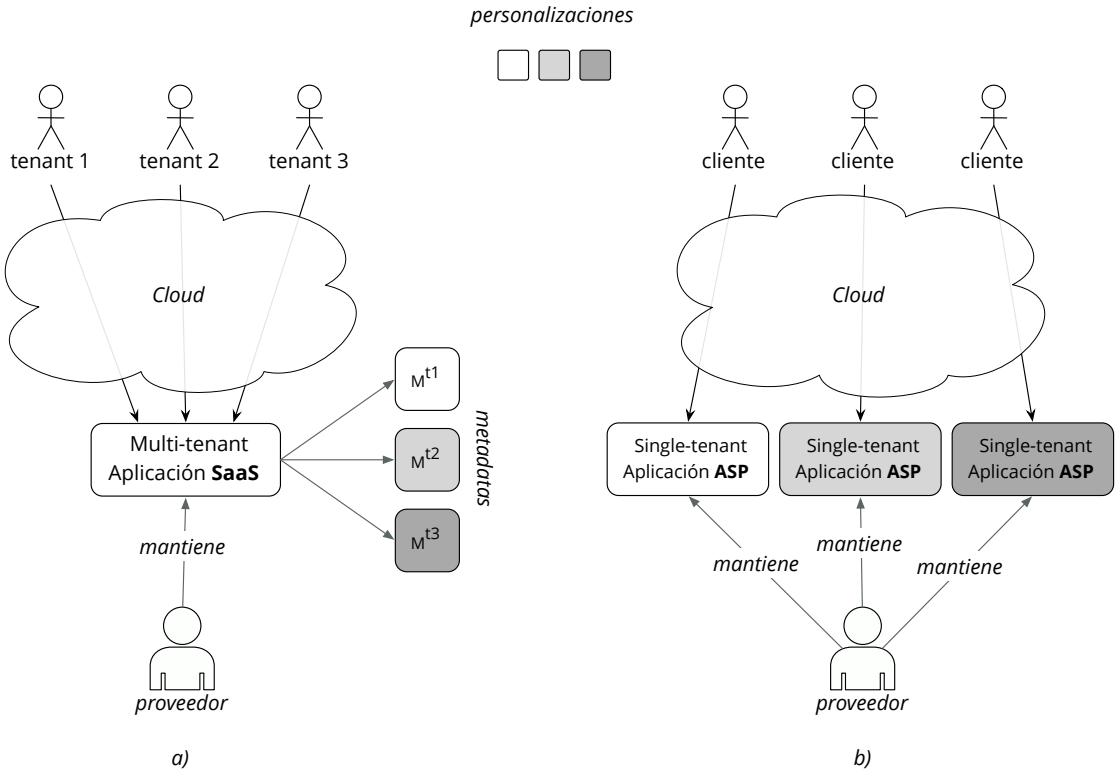


Figura 18. SaaS (a) vs. ASP (b). Single vs Multi tenant.

En la Figura 18 observamos como las aplicaciones SaaS pueden ser personalizadas gracias al uso de metadatos. Esta situación a permitido resolver uno de los principales problemas de ASP, al no tener que mantener tantas aplicaciones como clientes. El código ejecutado por los diferentes tenants es el mismo, por lo que los proveedores tienen la ventaja de difundir las actualizaciones de forma inmediata. Sin embargo complica el proceso de personalización; modificar el programa a nivel de código para adaptarse a las necesidades de cliente puede tener un efecto contrario y perjudicar al resto de suscriptores haciendo que el programa deje de ser útil para la organización.

El uso de metadatos permite a los tenants personalizar las aplicaciones sin necesidad de modificar que código de la aplicación. En SaaS la adaptación y personalización se consigue mediante una **configuración** de la aplicación mediante el uso de **metadatos**.

Es importante por tanto que las aplicaciones dispongan de una interfaz usable (un *wizard* por ejemplo) que permitan a los clientes modificar los metadatos para configurar las aplicaciones para adaptarlas a sus necesidades. Típicamente los usuarios pueden realizar cambios en la configuración en cuatro campos [13], [15], [39], [91], [92]:

- Interfaz de usuario e imagen corporativa.
- Flujo de trabajo y lógica de negocio
- Modelo de datos
- Control de acceso

En [93] se propone una técnica para conseguir un nuevo nivel de **personalización de backend**. Estas modificaciones permiten a los programadores inyectar lógica de negocio en las aplicaciones. De esta forma, la personalización la conseguimos directamente mediante la inserción controlada de código para cada tenant. En el capítulo relacionado con multi-tenancy estudiaremos los metadatos con mayor detalle.

6.5 Escalabilidad

La escalabilidad se refiere a la capacidad que tiene una aplicación de incorporar más datos en la aplicación o un mayor número de usuarios. Esto es sin duda más importante aún en SaaS debido a su entorno compartido ya que el número de usuarios finales se multiplica por el número de clientes/empresas suscritas a la aplicación [23].

Con el objetivo de dar soporte a un mayor número de usuarios, los programas SaaS pueden ser *escaladas hacia arriba* (*scale-up*) cambiando el servidor por otro más potente, mayor memoria y rapidez de unidades de disco o *escaladas hacia fuera* (*scale-out*) distribuyendo el procesamiento en diferentes servidores.

Volviendo a la comparación SaaS y ASP, ASP carece de escalabilidad ya que los proveedores software tienen que mantener tantas aplicaciones como clientes; cada uno con sus particularidades debido a las personalizaciones y adaptaciones necesarias. Inversamente, el modelo SaaS con su arquitectura multi-tenant subyacente presenta un alto grado de escalabilidad y permite la personalización y adaptación a nivel de tenant gracias al uso de metadatos. Los proveedores reducen costes de operativa al tener simplemente una aplicación que mantener.

6.6 Disponibilidad del servicio

La disponibilidad es el hecho de que un sistema se encuentre online y pueda ser accedido por sus usuarios. Un sistema no se encuentra disponible si los usuarios no pueden acceder al el debido a fallos en el hardware, sistema operativo o en la aplicación. Este tiempo de indisponibilidad generalmente se conoce como "*downtime*" y puede ser ocasionado por varios factures, desde mantenimientos planificados hasta fallos de infraestructura provocados por otros desastres [89]. Al resto del tiempo en el que el sistema se encuentra disponible y funcionando correctamente se le conoce como "*uptime*".

Uno de los factores que las empresas tienen en cuenta a la hora de sopesar posibilidades es precisamente la disponibilidad del servicio, ponderando las medidas de downtime que varían de unos proveedores a otros [94]. Para una empresa, el hecho de que un software no se encuentre disponible para su acceso puede ser muy costoso pudiendo suponer pérdidas de 4500$ por hora dependiendo del tipo de negocio [91].

La disponibilidad de servicio y la fiabilidad es quizá los mayores problemas a los que se enfrenta no SaaS, sino cloud computing en general. Las compañías proveedoras lo saben y es uno de los aspectos más trabajados. AWS presenta en su web del producto S3 la siguiente información [56]: *"Diseñado para una capacidad de duración del*

99.999999999 % y una disponibilidad del 99,99 % para los objetos durante un año dado."

En el momento de la contratación del servicio, los clientes también tienen que establecer acuerdos de compensación para el caso de fallo de disponibilidad o pérdidas de información [95].

6.7 Acuerdos de nivel de servicio (SLAs)

Los documentos de *acuerdo de nivel de servicio* (SLAs, *Service Level agreements*) definen la relación entre las dos partes: el proveedor y el receptor [96]. Bien usados, los SLAs sirven para:

- Identificar y definir las necesidades de cliente
- Establecer unas bases para el entendimiento
- Simplificar asuntos complejos
- Reducir áreas de conflicto
- Facilitar y promover el diálogo en caso de disputas
- Eliminar expectativas poco realistas

En su documento oficial de definición de estándares [90], el NIST considera que son muchas las razones por las cuales se deben fijar para el rendimiento. Los clientes necesitan ser capaces de determinar objetivamente los costes y beneficios de trasladarse a la nube; validando la información de rendimiento que reclaman los proveedores y pudiendo comparar también de manera objetiva a diferentes vendedores a fin de conseguir el servicio que mejor se adapte a sus necesidades. Aunque no sea una lista extensa, algunos aspectos de rendimiento relevantes son:

- Rendimiento en la gestión
- Comparativas de rendimiento
- Rendimiento de elementos relacionados con el ciclo de vida del servicio
    - Negociación
    - Instanciación
    - Terminación
- Pruebas de rendimiento

- Monitorización
- Auditoría

Los clientes y proveedores SaaS deben usar estos estándares y métricas como base para configurar un contrato a nivel de servicio (SLA) medible y aplicable.

Los SLAs son acuerdos contractuales legales y su no cumplimiento puede conllevar grandes pérdidas para los proveedores, en este caso los vendedores SaaS [23]. Dado que SaaS implica una migración de responsabilidades IT, los SLAs pueden ser utilizados por los clientes para medir el nivel de servicio esperado. Típicamente el contenido de un documento SLA se divide en [96]:

- Definición de servicios
- Gestión de rendimiento
- Gestión de problemas
- Obligaciones y responsabilidades del cliente
- Deberes y responsabilidades del cliente
- Garantías y soluciones
- Seguridad
- Recuperación de desastres
- Expiración del servicio

La existencia de SLAs permite a lo clientes tener un mayor control sobre los proveedores [88]. En la fórmula de pago por uso, los clientes abonan cuotas de forma recurrente durante la duración del contrato. Los proveedores SaaS están sujetos normalmente a SLAs mensuales y están financieramente motivados para mantener un correcto soporte y mantenimiento. En contraposición al software tradicional donde se pagan licencias perpetuas una sola vez, en SaaS los proveedores deben de tener una mayor disposición y responsabilidad con los clientes.

Si los niveles de calidad de servicio no cumple con los SLAs reflejados en el contrato, entonces los clientes pueden presionar con mayor fuerza legal a los proveedores

Según [89], hoy en día, muchos proveedores SaaS no tienen acuerdos formales de este tipo con sus clientes. Especialmente en el caso de aplicaciones del mercado intermedio, en donde el 80% de los proveedores no establecen contrato alguno con sus clientes. En cambio, los proveedores con targets de cliente de grandes compañías si suelen hacer uso de SLAs formales a la hora de suministrar el servicio.

6.8 Integración con otros sistemas, interoperabilidad y portabilidad

La **integración de sistemas** es el proceso por el cual diferentes sistemas y aplicaciones software son enlazados física o funcionalmente. Son estrategias y métodos para combinar un conjunto de sistemas interdependientes en uno sólo, permitiendo por tanto que dos o más aplicaciones interacciones e intercambien información de forma transparente [89]. La integración es un aspecto fundamental en los sistemas empresariales y la mayor parte de los autores consideran este aspecto fundamental en su definición

La **interoperabilidad** se refiere a la capacidad en que los datos pueden ser procesados por diferentes servicios en diferentes plataformas cloud usando especificaciones comunes.

La **portabilidad** debe ser considerada a dos niveles:

- *A nivel de datos*: La información pueden ser trasladada de un proveedor a otro y que las aplicaciones. Las aplicaciones deben de ser capaces de importar y exportar datos de manera eficiente y segura.

- *A nivel de aplicación*: las aplicaciones al completo pueden ser migradas a otro proveedor en otro centro de datos y además a un coste aceptable.

Como parte de la implementación de una aplicación SaaS, los datos deben de ser transferidos entre los sistemas existentes. Esto lo podemos conseguir siguiendo alguno de los siguientes modelos [89]:

- Importar al nuevo sistema la información de los sistemas antiguos (importación y exportación, portabilidad)

- Configurar la aplicación SaaS para que funcione con la información de los sistemas tradicionales on-premises anteriores (importación, interoperabilidad)

- Configurar los sistemas tradicionales para que usen la información extraída del nuevo sistema SaaS (exportación, interoperabilidad)

6.9 Accesibilidad

El objetivo de la accesibilidad es conseguir que las personas con discapacidad puedan acceder a la información y los datos del mismo modo que lo hacen las personas sin discapacidad. La accesibilidad es relevante a nivel de SaaS en donde los usuarios interaccionan con las aplicaciones. En esta interacción es donde los criterios de accesibilidad son medibles y muchos de los estándares fijados para las aplicaciones tradicionales son aplicables para SaaS [90].

En SaaS, las aplicaciones están basadas en web. Las pautas de accesibilidad web exponen cómo desarrollar los sitios web para hacerlos universales, accesibles a todo el mundo independientemente de su discapacidad. Estos han sido definidos por la *Iniciativa de Accesibilidad en la Web* (WAI) [97] del consorcio W3C. Según podemos leer en su página web:

La accesibilidad web significa que personas con algún tipo de discapacidad van a poder hacer uso de la web. En concreto, al hablar de accesibilidad web se está haciendo referencia a un diseño web que va a permitir que estas personas puedan percibir, entender, navegar e interactuar con la web, aportando a su vez contenidos. La accesibilidad web también beneficia a otras personas, incluyendo personas de edad avanzada que han visto mermadas sus habilidad a consecuencia de la edad.

Para un desarrollador/diseñador web, es importante entender cuál es realmente la causa y el efecto de las pautas que marcan la WAI.

6.10 The Long Tail

Las ventajas de SaaS, mucho más que económicas, no son sólo para el cliente, sino también para los proveedores del servicio. Estas empresas ven cómo pueden acceder a un mercado más amplio, a nichos de empresas que hasta ahora con el modelo

tradicional no era posible cubrir. Esta nueva cuota de mercado, conocida como *Long Tail* (larga estela o cola larga) está basada en el concepto de *"vender pocas cosas, pero a muchos clientes"* [22], [78].

La larga estela es el nombre coloquial para una bien conocida característica de las distribuciones estadísticas (*Zipf*, Ley de potencias, distribuciones de Pareto y en general distribuciones de *Lévy*). La característica es también conocida como *heavy tails, power-law tails*, o las *colas de Pareto*. Estas distribuciones son semejantes al gráfico mostrado en la Figura 19 [98]**.**

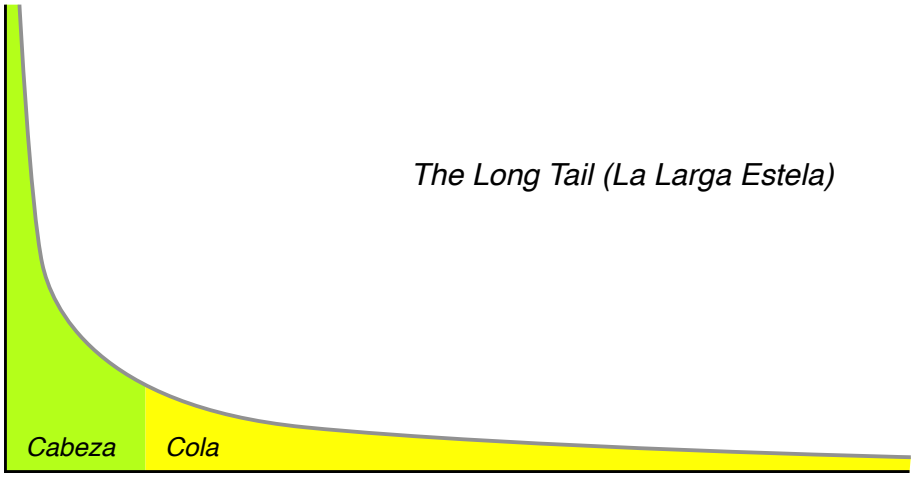


Figura 19. La larga estela (cola), puede comprender un área incluso mayor que la de la cabeza

En el artículo de la revista *Wired "The Long Tail" (La Larga Estela)* [78], Chris Anderson popularizó el término al explicar el éxito de Amazon en la venta de un mercado para el que los minoristas no están preparados. El modelo SaaS permite acceder a cuotas de mercado aplicando la máxima de "Vender menos, pero de todo".

Internet y el entorno digital han cambiado las leyes de distribución y las reglas del mercado. Para Anderson, la reducción del coste de almacenamiento y distribución permiten que ya no sea necesario focalizar el negocio en pocos productos. Ahora existen dos mercados:

1. **El mercado de masas**: centrado en el alto rendimiento de pocos productos y que según Chris Anderson ya se está quedando atrás.
2. **El nicho de mercados**: se basa en la suma o acumulación de todas las pequeñas ventas de muchos productos, que pueden igualar o superar al primero.

Ambos mercados son los que se encuentran representados en la Figura 19 y son la cabeza y la cola del gráfico, respectivamente.

La cantidad de nuevos discos, libros, DVDs que anualmente salen en el mercado son prácticamente infinitos. Solamente unos pocos de estos lanzamientos alcanzarán el nivel de best sellers. Los minoristas de "cemento y ladrillo" [23] tradicionales hacen normalmente acopio de estos productos de gran demanda, debido a que no tienen capacidad suficiente para abarcar el mercado completo, por escasez sobre todo de almacenamiento y el riesgo de aprovisionamiento. Los vendedores on-line, sin embargo no tienen estas limitaciones. Cuando un cliente realiza un pedido, los productos son enviados directamente desde grandes almacenes situados en todo el mundo. Pueden vender y publicitar con la misma facilidad, títulos de gran éxito en ventas, como otros menos populares (Figura 20 [23]). De esta forma, el almacenamiento se convierte en algo virtual además de no tener que aprovisionar por adelantado los productos, se sirve por demanda. El éxito de la venta al Long Tail consiste en la eliminación de costes por almacenamiento y provisión.

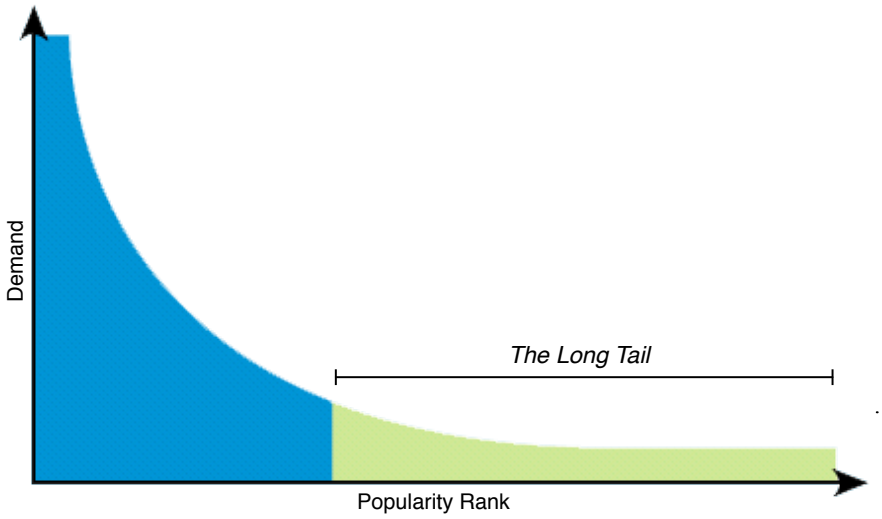


Figura 20. La estela está formada por productos de menor popularidad

Una gran empresa tradicional de venta de libros puede almacenar en sus estanterías sobre unos 130.000 títulos. Según Anderson, la mayor parte de las ventas de Amazon, no salen de esos 130.000 títulos, es decir, una empresa tradicional no tendría los libros que suele vender Amazon.

"Olvídate de hacerte rico gracias a unos pocos éxitos en los 40 principales. El futuro de la industria discográfica está en millones de nichos con pocos fans". Chris Anderson *[99]*

Esta idea original de *Chris Anderson*, fue posteriormente estudiada por *Chong* en su artículo *"Architecture Strategies for Catching the Long Tail What Is Software as a Service ?"* [23] y en él se explica con detalle este concepto y su relación con SaaS.

La necesidad de sistemas de información es también para las pequeñas empresas, y por cada gran empresa que compra el sistema, existen cientos de pymes que se podrían beneficiar de su uso.

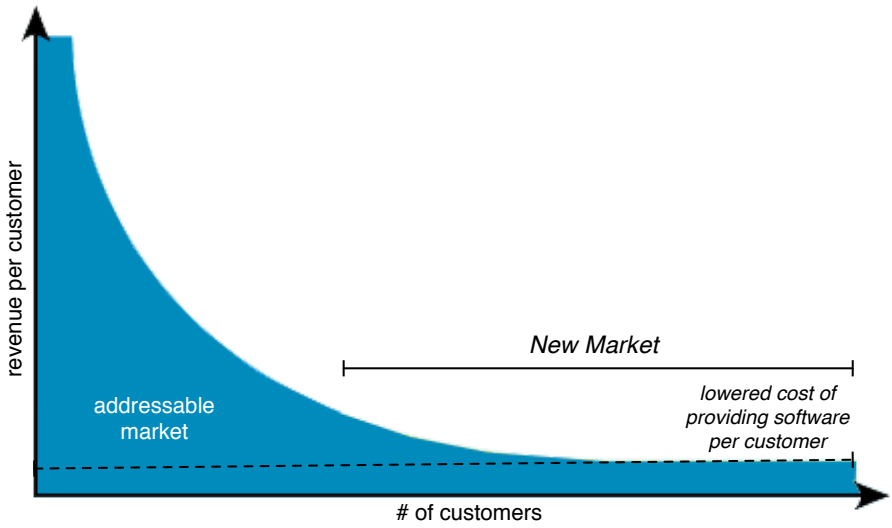


Figura 21. Nuevo nicho de mercado. El Long Tail de las pymes

Según vemos en la Figura 21, la reducción de costes que supone SaaS permite a sus proveedores aprovechar las economías de escala para ofrecer las soluciones a un precio más asumible; cosa que no pueden hacer los vendedores tradicionales. En el modelo anterior estas empresas no podían asumir previos tan elevados. Sin embargo, con esta nueva fórmula de distribución de costes, los pequeños clientes si podrían acceder suponiendo un nuevo nicho de mercado para los proveedores.

7 Ventajas y desventajas

En [100] se hace un guiño a la famosa frase de William Shakespeare y cita en su titular: *SaaS o no SaaS, he ahí la cuestión (To SaaS or not to SaaS?, that is the question).*

Efectivamente, el uso de SaaS trae muchos beneficios, pero también puede llevar inconvenientes [7], [84]. Las empresas deben ponderar estas antes de tomar una decisión. Hagamos un repaso de las mismas.

7.1 Beneficios

- **Menos inversión inicial y transferencia del riesgo**. El poder utilizar el software sin tener que realizar una inversión inicial en máquinas, software base (Sistema Operativo) y software adicional es un beneficio importante para los directores de IT y en definitiva para la empresa.

- **Reducción de costes**. Además de pagar solo por aquello que necesitamos, se obtiene un ahorro de costes de mantenimiento de servidor, estaciones de trabajo y del software necesario para el correcto funcionamiento de la aplicación. Además, al externalizar infraestructura también reducimos la necesidad de personal destinado a su mantenimiento.

- **Actualizaciones y nuevas funcionalidades inmediatas**. Aparte de prescindir de personal dedicado a la actualizaciones, dispondremos de actualizaciones las mismas incluyendo mejoras del software de manera inmediata.

- **Soporte más ágil y rápido.** Los bugs de la aplicación tienen un tratamiento directo y una puesta en servicio más rápida que en instalaciones locales.

- **Calidad del servicio.** Las aplicaciones SaaS están diseñadas para ser altamente escalables y poder dar servicio a miles de usuarios. Por consiguiente, las soluciones debe de estar implantadas en una infraestructura con tecnologías que permitan dicha carga, como por ejemplo el *clustering* [101].

- **La empresa centra su esfuerzos en su negocio.** Las empresas realmente externalizan los sistemas hasta el punto de no dedicar esfuerzos en la elección y mantenimiento de los mismos. Siempre requerirán atención del departamento IT pero en mucha menor medida que las soluciones tradicionales.

- **Mayor disponibilidad y seguridad de los datos**. En contra de lo que puede parecer y a la vista de las desventajas que veremos a continuación, muchas empresas no disponen de procedimientos de respaldo y restauración, y en general de planes de contingencia en caso de perdida de información o de fallo del hardware.

- **Mayor responsabilidad del proveedor.** La mayoría de la empresas que ofrecen software como servicio incluyen SLAs a medida para cada tipo de usuario. De este modo, los clientes SaaS pueden ejercer un mayor control sobre los proveedores que los clientes de software tradicional. Mientras que los proveedores SaaS están sujetos a los SLAs, en el software tradicional una vez realizada la venta, poca responsabilidad tienen los vendedores [88].

- **Ubicuidad *360º/365/24***. El acceso a la aplicación se realiza por la web, por tanto el uso del sistema es independiente de la localización y del horario. Los usuarios finales pueden ejecutar el sistema desde cualquier parte del globo (360º), cualquier día (365 días) y a cualquier hora (24 horas).

- **Ecología.** La etiqueta de "empresa ecológica" y responsabilidad con el medio ambiente mejora la imagen de las empresas. En una reciente encuesta [24] se descubrió que el 17% de las empresas con esta consideración aumentan el número de clientes, el 31% la fidelización y el 69% la satisfacción del cliente.

Desde el punto de vista de los proveedores, SaaS supone:

- **Reducción de costes.** El nuevo modelo de distribución reduce los costes de difusión por los canales tradicionales de marketing [85]. Además, los gastos de distribución y almacén desaparecen al no necesitar programas de instalación. A su vez, la producción, diseño y despliegue de software se reduce drásticamente debido a la arquitectura multi-tenancy subyacente. En contraposición, el grado de personalización de la aplicación es menor [88].

- **Aumento del espectro de clientes**. El acceso a la aplicación a través de la web y las facilidades en el pago (tarjeta de crédito, PayPal, etc.), permite incrementar el abanico de clientes potenciales de las empresas proveedores de software. No es

necesaria una distribución tradicional ni almacenamiento. Se abren nuevos nichos de mercado al tener la posibilidad de acceder al *Long Tail* [22], [78], [102].

- **Evitar la piratería**. En el modelo tradicional el software puede ser copiado de forma sencilla y a un coste prácticamente nulo; de este modo evitar la piratería es prácticamente imposible [85]. En SaaS los programas están implantados en data centers controlados por los proveedores que además tienen un mayor nivel de conocimiento en seguridad informática. De este modo la piratería es más complicada de conseguir y se necesitan más conocimientos que un mero programa de copiado.

7.2 Inconvenientes

- **Nivel de confianza bajo en la seguridad de los datos**. El hecho de que datos de la empresa (que pueden ser críticos o no pero que evidentemente son privados) no estén localizados dentro de las paredes de la empresa es algo que en general no suele gustar y sobretodo a la alta dirección que en determinadas ocasiones es conservadora y escéptica. *"Los datos confidenciales de mi empresa jamás estarán en la nube"* es la frase que muchos de los CEO comentan antes de rechazar la migración. Queda un trabajo arduo de convencimiento y exposición de la idea por parte del responsable de IT al resto de la directiva. El hecho de que los datos no estén en nuestra empresa no quiere decir que estén menos seguros. En el capítulo de multi-tenancy veremos como existen *técnicas de cifrado* que aumentan considerablemente la seguridad de nuestros datos [103][104].

- **Integración con el resto de la aplicaciones y sistemas**. Como lo normal será tener aplicaciones con instalación local con distintas aplicaciones SaaS, existe un aumento de la complejidad en el caso de que queramos conectar o explotar los datos que tenemos en la nube con los datos que tenemos en la empresa. Esto aumenta el grado de importancia a medida que los datos que mantenemos en la nube sean importantes.

- **Migración de datos la nube**. Si además de la desventaja anterior la aplicación o plataforma no dispone de un sistema (Web Service, API, etc.) que permita extraer

los datos, nos enfrentamos claro inconveniente para no implantar la aplicación en la nube.

- **Disponibilidad del servicio.** Guarda relación con el grado de confianza que tengamos sobre el proveedor del software o plataforma. Las empresas consideran que su correcto funcionamiento dependerá de buen funcionamiento del proveedor, problema que no tendrían en caso de tener la aplicación instalada localmente. Ponemos una aplicación crítica para nuestra empresa en manos de un proveedor externo. Por otro lado, un fallo puede ser fatal en la reputación de un proveedor; Amazon Web Services tuvo el 7 de Agosto de 2011 uno de sus cortes más sentidos. Un rayo cayó sobre el CPD de Amazon en Irlanda y se llevó por delante empresas de nombre como: PayPal, Microsoft, Amazon, NeoTeo, Filmin o Menéame [105].

- **Dificultad de encontrar aplicaciones especializadas.** SaaS no está disponible para determinadas aplicaciones de una industria específica.

Capítulo III

Multi-tenancy

Multi-tenancy es un patrón arquitectónico de aplicaciones cloud y pieza clave para el éxito de SaaS. Este capítulo de la tesis estudia el concepto en detalle tanto por su importancia, como por suponer la base de nuestra propuesta. Tras una breve introducción y definición, se describe la arquitectura de este tipo de sistemas. Posteriormente, se analiza el estado del arte para la capa de datos, exponiendo los enfoques existentes así como los factores para su elección y técnicas de escalado.

1 Introducción

Cloud computing permite que todo el mundo pueda acceder a grandes capacidades computacionales [31]. La computación se suministra como un servicio por parte de los proveedores cloud [4], mientras que las empresas desarrolladoras hacen uso de ese acceso para ofertar a su vez las aplicaciones a sus clientes. Este nuevo paradigma ha cambiado la fórmula de distribución de software, que ha pasado de comprarse, a pagar por su uso (SaaS, Software as a Service). Los usuarios pasan de ser propietarios a "arrendatarios" (*tenants*) de las aplicaciones pagando bajo demanda el uso de las mismas.

Este nuevo modelo SaaS favorece sobre todo a las pequeñas y medianas empresas ya que, además de precios reducidos, pueden acceder a aplicaciones de calidad hasta entonces reservadas para las grandes corporaciones. Los proveedores por su parte, también se ven beneficiados debido a que aumentan su mercado potencial, pudiendo ofertar el servicio a pequeñas y medianas empresas.

En entornos cloud, **la compartición de recursos es clave no sólo a nivel de infraestructura, sino también a nivel software**. Las aplicaciones deben de ser utilizadas por distintos clientes para que la reducción de costes se haga efectiva y aumenten por tanto el número de usuarios y el negocio de los proveedores [16].

Las Arquitecturas multi-tenant (AMTs) posibilitan que varios clientes compartan la aplicación y por tanto los costes de uso, que se ven reducidos. El modelo multi-tenant (MT) es considerado como una característica esencial en cloud computing y su fórmula SaaS para la distribución del software [12], [33], [103], [106].

Salesforce.com, uno de los proveedores cloud más populares no considera SaaS las soluciones que no usan multi-tenancy [106]: *"los modelos de alojamiento que no aprovechan las ventajas de multi-tenancy no pueden discutirse dentro de la misma propuesta de valor que supone el término SaaS"*.

El uso de **metadatos** permite a las empresas personalizar y configurar las aplicaciones Gracias a ellos, los tenants pueden modificar la presentación del sistema, modificar el modelo de datos o trazar cambios en los flujos de trabajo. Todo ello manteniendo el código de una aplicación MT invariable sin que el resto de tenants perciban los

cambios. Gracias a los metadatos, multi-tenancy pasa desapercibido para los diferentes suscriptores que tienen la impresión de disponer un **programa dedicado y configurado a medida** [107].

Una de las principales preocupaciones para la adopción de aplicaciones en entornos compartidos es la privacidad de los datos y la seguridad [26], [108], [109]. La confianza en el sistema depende de ellos y cualquier brecha puede provocar que la aplicación no se siga usando, además de minar la reputación del proveedor y otros perjuicios legales. Debido a esta situación, **la capa de datos en estas arquitecturas es esencial** y ha traído gran atención entre los académicos. Para ganar la confianza del cliente, una de las mayores prioridades de los arquitectos SaaS es la creación de diseño de robusto y seguro de la capa de datos que elimine las preocupaciones de aquellos clientes temerosos de dejar en manos de terceros el control de su negocio [103].

2 ¿Qué es Multi-tenancy?

La palabra *multi-tenancy* procede del inglés y quiere decir *multi-arrendatario,* esto es, cuando varios inquilinos hacen uso del mismo servicio. Aplicando este modelo a SaaS, podemos **agrupar varios clientes** para que compartan el uso de una misma aplicación y se reduzcan los costes tanto a nivel de hardware como a nivel de software. Algunas traducciones lo interpretan como "*multi-propietario*". En esta tesis, además de considerar no acertada esta traducción (en el modelo SaaS no podemos hablar de "propietarios" reales), nos limitaremos a usar anglicismos como tal, sin traducción.

Un ***tenant*** es una unidad organizativa que hace uso de la aplicación SaaS en una modalidad de pago y condiciones regidas por un contrato de suscripción. Un tenant puede estar formado por muchos usuarios finales y por tanto las aplicaciones MT son también multi-usuario. Llamaremos ***tenancy*** *(tenancies* en plural*)* al conjunto de usuarios finales de un mismo tenant que ejecutan la misma visión personalizada de la instancia de aplicación. El número de instancias de la aplicación puede ser más de uno dando como resultado un ***multi-tenancy farm*** [15]. Esta situación puede ocurrir no sólo por motivos de balanceo de carga sino que también por cuestiones legales. Por ejemplo existen regulaciones en determinados países que obligan a las entidades a almacenar los datos dentro de las fronteras del mismo territorio [29], [39].

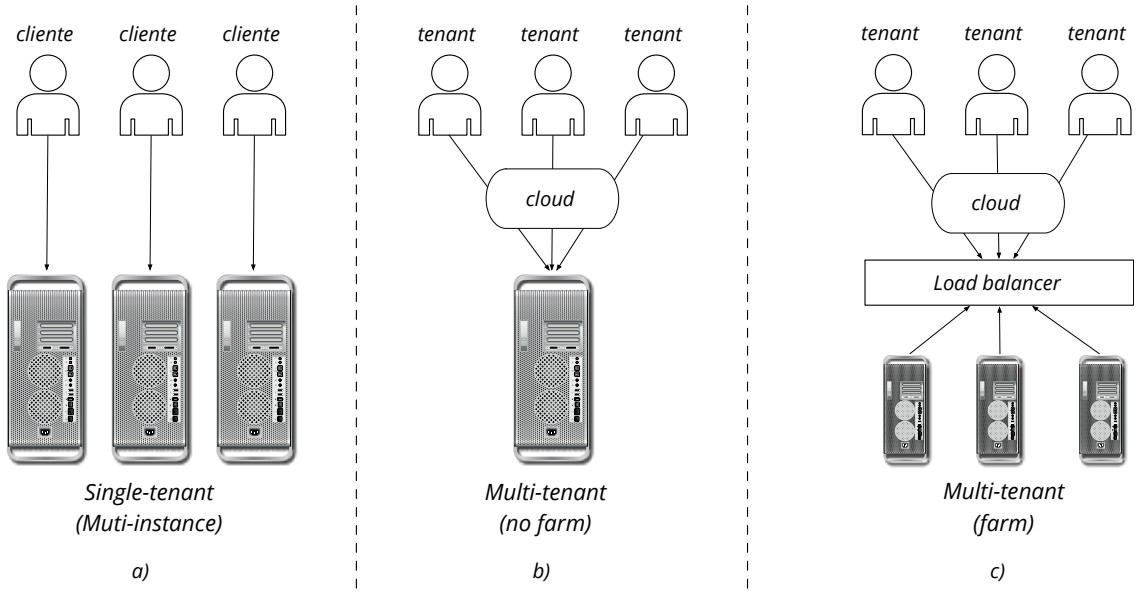


Figura 22. Arquitecturas single tenant (a), multi-tenant sin farm (b) y multi-tenant con farm (c)

La Figura 22 extiende a la vista anteriormente en el capítulo de SaaS. En ella podemos ver una comparativa de las arquitecturas single-tenant, en la que existe una instancia de aplicación por usuario y los entornos compartidos multi-tenant. En este caso añadimos la posibilidad de repartir a los arrendatarios en farm balanceado de servidores.

Así, un entorno *multi-tenant* típico alberga diferentes clientes que comparten un mismo software, que se ejecuta en la misma máquina, con el mismo sistema operativo y sistema de gestión de bases de datos. **La distinción entre un cliente y otro se realiza durante la fase del diseño del software**, gracias a modelos de arquitectura que aseguren la privacidad de los datos y que articulen un uso simultáneo transparente, de tal forma que los clientes no tienen conciencia acerca de que el entorno es compartido. La arquitectura multi-tenant varía de la *multi-instance* en que esta última hace uso de implantaciones diferentes del mismo software para cada uno de los clientes.

Este uso compartido que propone multi-tenancy puede ser conseguido también a nivel de sistema mediante tecnologías como virtualización [110], [111]. Sin embargo, las máquinas virtuales establecen límites lógicos entre los recursos de los tenants; una separación que afecta a la eficiencia del sistema [91] y que finalmente termina incurriendo en nuevos costes. Además, la implementación de multi-tenancy por virtualización no es capaz de atender a la demanda [110] e impone a los sistemas un número mucho más bajo en el número de tenants posibles a alojar [39]. En este capítulo

discutiremos el nivel de aplicación donde multi-tenancy se consigue a través de la arquitectura software. Con este modelo el uso del servidor es más eficiente y los costes se rebajan. Sin embargo, no todo son ventajas ya que los tenants exigentes puede que consuman muchos recursos afectando al resto de tenants y comprometiendo por ende el rendimiento general del sistema. Esta situación se puede resolver empleando farms para el equilibrio de carga y/o aplicando otros métodos de aislamiento de rendimiento [110].

En una aplicación con arquitectura MT, los diferentes tenants que acceden al sistema son gestionados por el proveedor gracias a un módulo administrativo de la arquitectura. Este "*framework administrativo*" [15] no solamente debe gestionar las cuentas de cliente, sino que también debe de ser capaz de escalarlos incluyéndolos en el mismo servidor (*scale up*) o sacándolos fuera (*scale out*) a otro dentro del farm.

En resumen:

Multi-tenancy es un patrón de arquitectura para el modelo SaaS que surge como consecuencia natural a la necesidad de aplicar economías en escala. Gracias a las *Arquitecturas Multi-tenant (AMTs)*, varios clientes (tenants) comparten de manera transparente una sola instancia de la aplicación. A través de los metadatos, los tenants tienen la impresión de acceder a un software único y personalizado para su organización.

Con multi-tenancy obtienen beneficio todos los actores involucrados en cloud computing. Los vendedores cloud sirven capacidades computacionales a las empresas desarrolladoras; estas a su vez son proveedores SaaS que obtienen beneficios de un modelo de distribución de software al poder acceder al mercado del *Long Tail* [22] de las pequeñas y medianas empresas (clientes SaaS) que ya pueden soportar el precio de la subscripción.

3 Arquitecturas multi-tenant

Las arquitecturas software describen un conjunto de componentes del sistema así como sus relaciones topológicas [10]. Una importante contribución al campo de las arquitecturas de aplicación es la **definición de patrones** que puedan ser usados como guías y planos a la hora de diseñar por ejemplo un sistema de información empresarial [11].

Teniendo en cuenta lo anterior y recordando la definición prestada a inicios del capítulo definimos multi-tenancy como un **patrón de arquitecturas software** que permite que varios clientes (tenants) compartan la misma instancia de una aplicación software [91].

3.1 Madurez SaaS y niveles multi-tenancy

[23] considera que una buena aplicación SaaS debe integrar tres atributos: *escalabilidad, configurabilidad y eficiencia multi-tenant*. Partiendo del grado de presencia de los mismos propone un modelo de madurez SaaS en cuatro niveles, los cuales los cuales están resumidos en la Figura 23 [23].

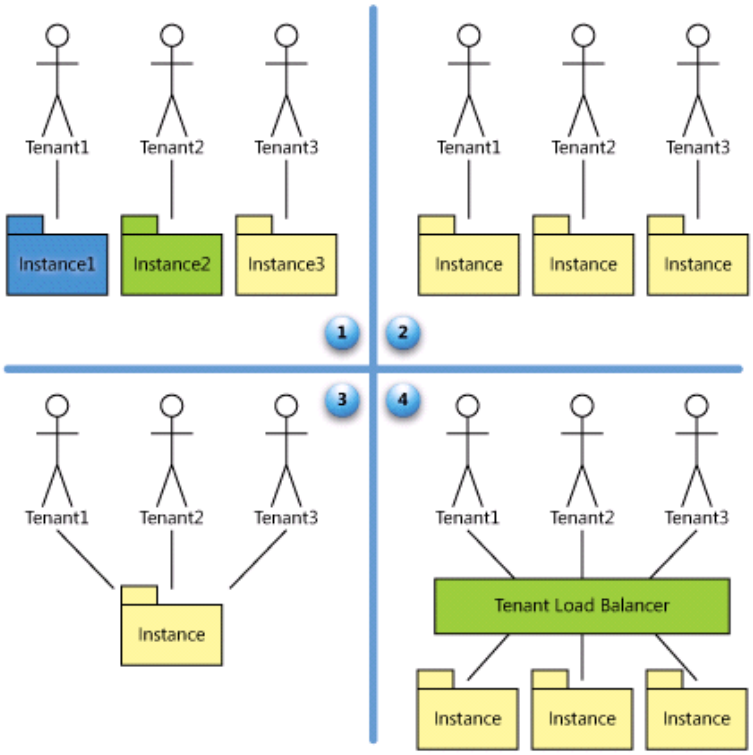


Figura 23. Niveles de madurez SaaS

3.1.1 Nivel 1: Ad Hoc (a medida)

El primer nivel de madurez es semejante al de las aplicaciones ASP de los años 90. Se trata de un modelo con arquitectura multi-instancia donde cada tenant dispone de su instancia particular de la aplicación. Cada aplicación individual está alojada en los servidores del proveedor y es personalizada a medida según las necesidades de cada cliente.

Normalmente, la conversión de las aplicaciones cliente-servidor tradicionales al nivel 1 de madurez es sencilla. Requiere poco esfuerzo en términos de desarrollo y no es preciso modificar la arquitectura en exceso. Aunque este nivel de madurez ofrece pocas de las ventajas que nos aportan niveles totalmente maduros de SaaS, permite al proveedor reducir costes al consolidar el hardware y el mantenimiento del servidor.

3.1.2 Nivel 2: Configurable

En este segundo nivel de madurez, tampoco existe soporte multi-tenancy. El proveedor de software mantiene una instancia de la aplicación por cada cliente. La diferencia radica en la configurabilidad de la aplicación. En el nivel 1, la aplicación es personalizada a medida para cada cliente, mientras que en este segundo nivel, el código de la aplicación es siempre el mismo. La personalización del software para cada cliente viene dado por unas **amplias posibilidades de parametrización**. Aunque las aplicaciones sean exactamente iguales, permanecen aisladas unas de otras (multi-instance).

La existencia de un único código de aplicación permite a los vendedores trasladar de manera rápida y efectiva las actualizaciones de la aplicación a los diferentes vendedores. Sin embargo, ya no es tan fácil la conversión de una aplicación tradicional a este nivel, al ser necesario una reestructuración de la arquitectura. Además, al igual que el primer nivel, las necesidades hardware y de almacenamiento son elevadas, ya que es necesario que el servidor proporcione servicio a múltiples instancias independientes.

3.1.3 Nivel 3: Configurable y multi-tenant

En el tercer nivel de madurez existe una única instancia de la aplicación que comparten todos los arrendatarios. La personalización de la aplicación se realiza gracias a los metadatos que permiten a los tenants configurar la aplicación. La seguridad y autenticación aseguran la privacidad de los datos, separando los que son de cada cliente y evitando accesos indeseables. Desde el punto de vista del usuario, la transparencia es absoluta y este tiene la impresión de estar trabajando en un software completamente a medida sin estar compartido con el resto.

Esta aproximación disminuye las necesidades de espacio, al estar compartido el código de la aplicación. El coste por ende, también disminuye y los recursos son gestionados de

manera más eficiente. La principal desventaja de este modelo es la escalabilidad. En este caso, solamente podremos hacer un *scale-up* de la aplicación al migrarla a un servidor más potente. La escalabilidad siempre estará limitada por la potencia del servidor en la que se encuentre alojada.

3.1.4 Nivel 4: Escalable, configurable y multi-tenant

En el último nivel de madurez, el proveedor proporciona servicio a múltiples clientes en una multi-tenancy farm de carga balanceada con instancias idénticas. De nuevo, la presencia de metadatos asegura a los tenants una experiencia única de usuario mientras que la información de cada tenant se mantienen segura y separada entre suscriptores. En este nivel se tiene una escalabilidad independiente de la potencia del servidor, al poder realizar un *scale-out* de clientes. En este nivel, el número servidores es arbitrario ya que depende del número de clientes y puede crecer o bajar según demanda.

3.2 Elección de un nivel de madurez.

Previamente a la comercialización de su aplicación, los proveedores deben de sopesar el nivel de madurez de la misma. Uno puede esperar que el cuarto nivel sea el objetivo último de cualquier aplicación SaaS, pero no siempre es el caso. Para ayudar a la elección sea quizás mejor pensar en el nivel de madurez como un continuo entre el aislamiento de los datos y el código en un lado y uso compartido en el otro (Figura 24 [23]).

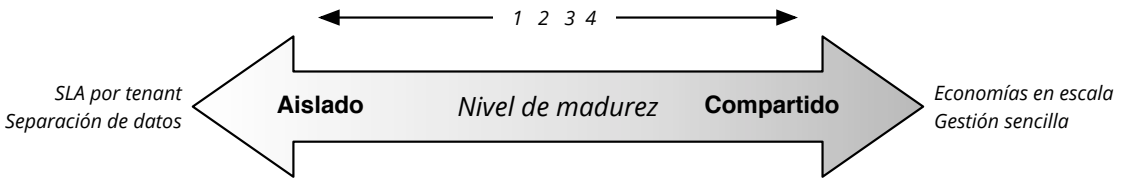


Figura 24. Separación continua de los niveles de madurez SaaS

Comercializar SaaS en un extremo u otro depende de las necesidades de negocio, arquitectura y operacionales, así como de los clientes. Analizamos a continuación cada uno de ellos.

3.2.1 Modelo de negocio

Optar por un modelo aislado elimina los beneficios económicos del compartido y ofrecer las aplicaciones a mayor coste. Sin embargo, existen circunstancias bajo las cuales pueda interesar este enfoque para cumplir con otros requerimientos. Además, los clientes pueden mostrarse contrarios por razones legales o culturales a tener los datos compartidos con otros tenants. Aunque se les diga que la privacidad de los mismos no correo peligro, al final lo importante es desarrollar un modelo de negocio que suponga beneficios.

3.2.2 Arquitectura

La aplicación puede simplemente no ser capaz de ejecutarse en un modelo centralizado multi-tenant con una sola instancia. Si por ejemplo estamos tratando de migrar una aplicación de escritorio con arquitectura cliente-servidor, hay incompatibilidades fundamentales al transformarla a una versión basada en web que no puedan superarse. En cambio, si programamos la aplicación desde cero tenemos mayor libertad de actuación y entonces podemos seleccionar un modelo compartido.

3.2.3 Necesidades operacionales

En ocasiones, lo modelos compartidos no son capaces de satisfacer los SLAs de los contratos. Para la selección de un nivel u otro, debemos de leer detenidamente las obligaciones establecidas en los contratos relacionadas con los tiempos de indisponibilidad del servicio, soporte y recuperación de desastres para determinar si estas obligaciones pueden ser satisfechas en un nivel compartido de aplicación

4 Arquitectura de alto nivel

Según [13], las aplicaciones SaaS son muy similares a otras basadas en arquitecturas orientadas a servicios (Service Oriented Architecture, SOA).

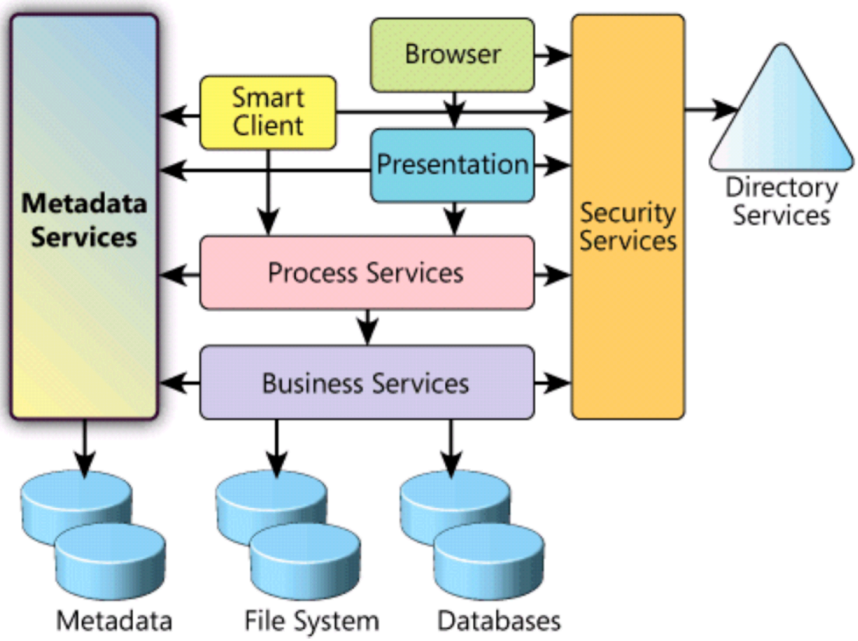


Figura 25. Arquitectura MT de alto nivel

La mayor parte de los componentes representados en la Figura 25 [13] deben ser familiares a los arquitectos software. Los servicios de procesamiento exponen la interfaz a la cual pueden llamar los distintos clientes (tanto *smart clients* como navegadores web). Estas llamadas generan flujos de trabajo o transacciones que a su vez demandan servicios al nivel de negocio, que es la que interacciona con la capa de datos y/o el sistema de archivos para almacenar o leer información. La capa de seguridad es la responsable de controlar el acceso a los usuarios finales y a los servicios software backend.

La principal diferencia con SOA es la existencia de una capa transversal de metadatos responsable de la gestión de la configuración individual de los tenants. Los *smart clients* y las capas de servicio pueden interaccionar con los metadatos para obtener las configuraciones personalizadas de cada tenant.

La seguridad es otro componente transversal de vital importancia. Si la privacidad es de per sé un requisito fundamental en cualquier aplicación multi-usuario, cobra mucha más envergadura en ambientes multi-tenant.

En las siguientes subsecciones pasamos a estudiar en detalle estos últimos componentes transversales, que hacen particulares a las AMTs, los metadatos de personalización y la seguridad.

4.1 Metadatos

Este componente transversal contiene aquellos servicios encargados de gestionar la configurabilidad de la aplicación para cada tenant.

Los servicios de metadatos permiten a los suscriptores SaaS personalizar las aplicaciones de acuerdo a sus necesidades. Esta particularización puede ser realizada a varios niveles, los cuales detallamos a continuación [15], [23], [39], [91], [92]:

- **Interfaz de usuario e imagen corporativa:** Posibilidad de modificar colores, gráficos, logos, etc. Esta habilidad es bastante apreciada por las empresas.

- **Flujo de trabajo y lógica de negocio**: Las aplicaciones SaaS deben ser capaces de establecer variaciones al flujo de trabajo; de este modo serán de utilidad para un gran número de organizaciones. Por ejemplo, una empresa puede que necesite que las facturas sean aprobadas por un responsable antes de ser emitidas; otra puede que necesite dos aprobaciones de forma secuencial y una tercera requerir una doble aprobación pero que pueda hacerse de forma paralela e indistinta. De esta forma el flujo de trabajo se alinea con sus procesos de negocio.

- **Modelo de datos**: Es imposible desarrollar un modelo de datos que se adapte a las necesidades de todas las empresas. Los tenants deben de ser capaces de *extender* el modelo de datos base de la aplicación para adaptarlo a sus necesidades. Más adelante estudiaremos en profundidad los técnicas de mapeo que permiten la extensión de la capa de datos.

- **Control de acceso**: Los tenants son responsables de la creación de las cuentas de usuario final, de determinar qué funcionalidades y recursos éstos pueden de acceder. Los derechos de acceso y restricciones están basados en políticas de seguridad que han de ser configuradas por cada tenant.

- En [93] se propone una técnica para conseguir un nuevo nivel de **personalización de backend**. Estas modificaciones permiten a los programadores inyectar lógica de negocio en las aplicaciones. De esta forma la personalización la conseguimos directamente mediante la inserción controlada de código para cada tenant.

4.2 Seguridad

Todo arquitecto SaaS debe considerar la seguridad como uno de los aspectos supremos en el diseño de las aplicaciones. Recordemos que estamos pidiendo a los clientes que de un cierto modo, nos den control sobre los datos de su negocio. Lo más seguro es que la aplicación SaaS empresarial maneje información comprometida de cada cliente. Además, a este hecho debemos unirle el uso compartido de la aplicación, factor que aumenta las exigencias con respecto a las aplicaciones no compartidas.

En una aplicación **multi-tenant**, el concepto de usuario final es diferente al de una aplicación tradicional. Debemos distinguir entre tenant y usuario. Desde este punto de vista podemos definir tres niveles de seguridad en una aplicación MT:

- **Nivel de suscripción**, hace referencia a que la aplicación no debe dar acceso a cuentas inexistentes o expiradas.
- **Nivel de tenant**, hace referencia a la privacidad de los datos de los diferentes tenants dentro del sistema MT.
- **Nivel de usuario final**, hace referencia al control que debemos llevar dentro de la aplicación para que, dentro de los datos de un tenant, podamos restringir el acceso a recursos.

Una aplicación SaaS segura tiene diferentes niveles de defensa complementarios que proporcionan protección a la capa de datos, tanto a nivel interno como externo (defensa en profundidad, *defense in depth*). En [23] se estudian en detalle una serie de factores

que se apoyan en tres características subyacentes para proporcionar la seguridad adecuada en el momento preciso:

- **Filtrado**: Uso de un componente para *transformación de consultas* como capa intermedia entre el tenant y la fuente de datos. Proporciona transparencia de tal forma que parece que solo sus datos son los que se encuentran en la base de datos.
- **Permisos**: Crear *listas de control de acceso* (ACLs, Access Control Lists) que determinen quién puede acceder a los datos y qué pueden hacer con ellos.
- **Encriptación**: Evita que los datos críticos estén disponibles para personal no autorizado aun cuando dispongan ya de ellos.

A continuación pasamos a detallar estos factores.

4.2.1 Conexiones fiables a la base de datos

En un entorno multiusuarios, los arquitectos de las aplicaciones usan tradicionalmente tres métodos para acceder a la capa de datos: *suplantación de identidad* (impersonation) y *cuenta de confianza del subsistema* (trusted subsystem account) y *acceso híbrido* (hybrid access) [103].

Con la **suplantación de identidad**, cada usuario accede con sus propios privilegios de acceso. La aplicación se presenta a la base de datos con las credenciales del usuario final, suplantando por tanto su identidad (Figura 26).

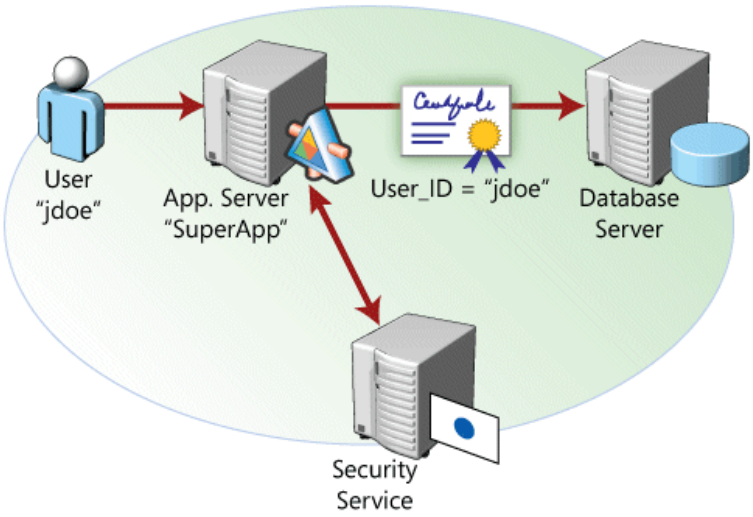


Figura 26. Suplantación de identidad.

En el segundo método de acceso (***cuenta de confianza de subsistema)***, la aplicación se conecta a la base de datos usando sus propias credenciales, independientemente del usuario final que se encuentre en la aplicación (Figura 27). Es necesario por tanto programar un nivel seguridad más a nivel de aplicación en que los usuarios no puedan acceder a determinados datos para los que no estén autorizados.

Para cada tenant, debe de existir al menos un usuario final con la **capacidad de definir roles y permisos** para el resto de usuarios finales, dentro de su tenancy.

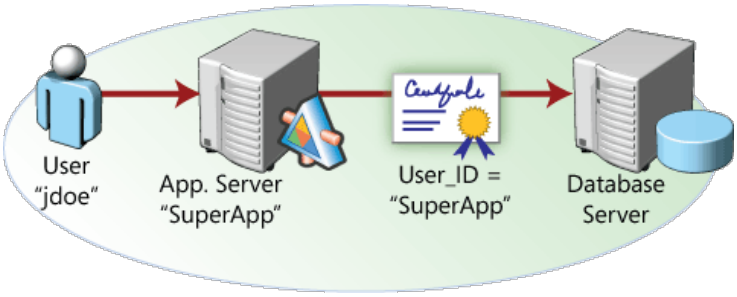


Figura 27. Cuenta de confianza de subsistema.

Teniendo en cuenta esta distinción, podemos definir un último método de **acceso híbrido**. Tal y como podemos ver en la Figura 28, en este método usamos suplantación de identidad a nivel de tenant, y una vez que encuadrado dentro de una tenancy, *trusted subsystem account*. Este enfoque implica la creación de cuentas de acceso a la base de datos para cada tenant, además del uso de ACLs que permitan a dichas cuentas acceder a los objetos de la base de datos pertinentes.

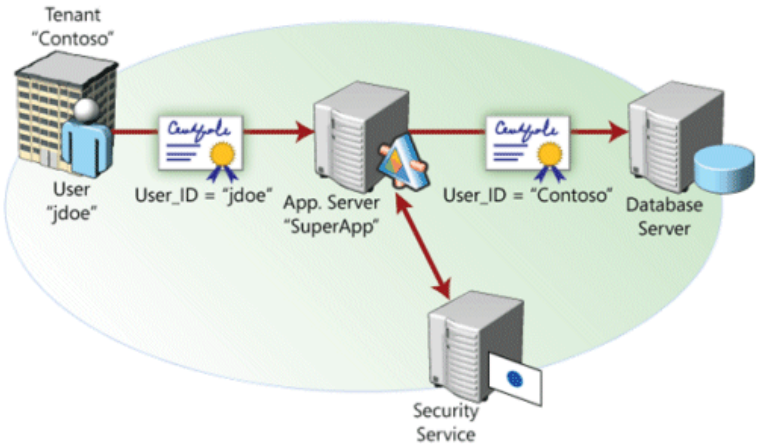


Figura 28. Acceso híbrido.

4.2.2 Tablas seguras

Podemos aumentar la seguridad a nivel de tabla en la base de datos mediante el uso del comando GRANT de SQL:

```
GRANT SELECT, UPDATE, INSERT, DELETE ON [TableName] FOR [UserName]
```

Este patrón es apropiado si almacenamos los datos de los clientes de forma aislada en bases de datos o en tablas separadas. En el primer caso, la seguridad puede ser implementada con la restricción a nivel de base de datos, pero también podemos aplicar este patrón como medida extra de seguridad.

4.2.3 Vistas/Filtros a nivel de tenant

En SQL, una vista es una tabla virtual definida con una sentencia SELECT que determina el filtro sobre los registros de la tabla. Las vistas SQL nos permiten restringir el acceso a determinadas filas de una tabla en la base de datos.

El uso de vistas como filtro a nivel de tenant en la base de datos supone una buena medida de seguridad extra sobre todo en **el enfoque compartido** de la arquitectura de la capa de datos.

Podemos crear una vista por cada tabla compartida que nos asegure el aislamiento virtual a nivel de tenant. Por ejemplo supongamos una aplicación CRM que tenga una tabla clientes; en nuestro vecindario existe un tenant con el TENANT_ID = 5.

```
CREATE VIEW CLIENTES_5 as SELECT * FROM CLIENTES WHERE TENANT_ID=5
```

De este modo, no solamente aumentamos la seguridad en el acceso compartido a nivel de tenant, sino que evitamos posibles brechas futuras ya que simplificamos el desarrollo por parte de los programadores de aplicaciones a terceros. Los programadores actuarán sobre una vista sobre la que pueden realizar consultas como si estuvieran en un entorno mono propietario, sin necesidad de dar a conocer la estructura real multi-tenant de la base de datos.

4.2.4 Encriptación de los datos de los tenants

Otro camino más de protección de datos consiste en el cifrado de la base de datos, de este modo, los datos permanecerán seguros aún cuando caigan en las manos incorrectas.

La importancia de la encriptación aumenta cuando más cercano esté nuestro diseño al modelo compartido de datos. En este modelo, los datos de usuario pueden encontrarse incluso dentro de la misma tabla, aumentando el riesgo de accesos no permitidos.

Los métodos criptográficos los podemos categorizar en *simétricos* y *asimétricos*. En **criptografía simétrica** la misma claves se usan tanto para encriptar como para desencriptar los datos (Figura 29). Por el contrario, en **criptografía asimétrica** (también llamada criptografía de llave pública) existen dos claves de cifrado: la llave pública y la llave privada. La información es codificada con la clave pública y descifrada con la clave privada. La clave pública es proporcionada a todas aquellas entidades de interés en comunicarse con el poseedor de la clave privada (Figura 30). De este modo, se aumenta el nivel de seguridad con respecto al modelo simétrico ya que, aunque se desvele la clave pública, el criptograma permanecerá seguro puesto que solamente el poseedor de la llave privada podrá descifrarlo.

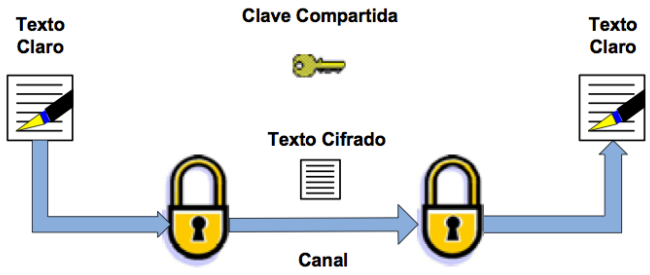


Figura 29. Criptografía simétrica

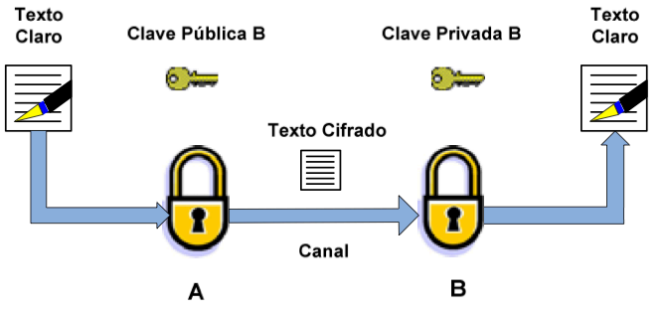


Figura 30. Criptografía asimétrica

La criptografía de llave pública requiere mayor capacidad de computación que la simétrica. En la actualidad, este esfuerzo pude verse traducido en cientos, e incluso miles de veces el tiempo necesario en la codificación y decodificación usando pares fuerte de claves asimétricas frente a una clave simétrica de la misma calidad.

Debido a estos altos requerimientos, este enfoque se hace prácticamente inviable para aplicaciones SaaS. Tenemos que codificar cada dato almacenado en la base de datos y el esfuerzo de procesamiento es demasiado exigente repercutiendo en el rendimiento. Un mejor enfoque consiste en utilizar un tercer método que contenga los beneficios de los dos anteriores: ***key wrapping.***

En este tercer método, para cada tenant son generadas tres claves de encriptación: una simétrica y dos pares asimétricos. La clave simétrica es usada para la encriptación de datos debido a su menor consumo de computación, mientras que el par asimétrico se utiliza para la codificar la clave simétrica.

Cuando un usuario final se conecta a la aplicación, el sistema utiliza suplantación de identidad para acceder a la capa de datos y obtener la clave privada. La aplicación puede ahora por tanto descodificar la clave simétrica que finalmente usará para leer y escribir en la base de datos.

Estamos por tanto ante otro ejemplo de defensa en profundidad en el acceso a los datos. Con la encriptación, una brecha que expusiera los datos no tendría consecuencias a no ser que se dispusiera de la clave simétrica que los descodifica. Dicha clave sólo la podremos tener codificada con cifrado asimétrico de alta seguridad.

5 Bases de datos multi-tenant

En escenarios compartidos como las aplicaciones MT, las bases de datos son un aspecto crucial ya que las empresas consideran la privacidad de los datos como sus principales preocupaciones en la adopción de SaaS [26].

5.1 Implementación de multi-tenancy

Son varios los autores que han investigado sobre diferentes enfoques. Aunque no emplean la misma terminología para aproximaciones semejantes, todos ellos coinciden en que **la distinción entre uno y otro viene dado por el aislamiento de los datos de cliente** [103]. Podemos considerar un enfoque **aislado** cuando los datos de los clientes se almacenan en lugares diferentes; por el contrario, un enfoque será **compartido** cuando los datos de los diferentes tenants se encuentren en el mismo lugar. Según apunta [15], [23], [57], [103], el nivel de aislamiento no presenta una frontera fija binaria, sino que la separación entre una tipología u otra es continua (Figura 31). Esto es, los enfoques no son aislados o compartidos y pueden estar posicionados en cualquier lugar entre ambos extremos.

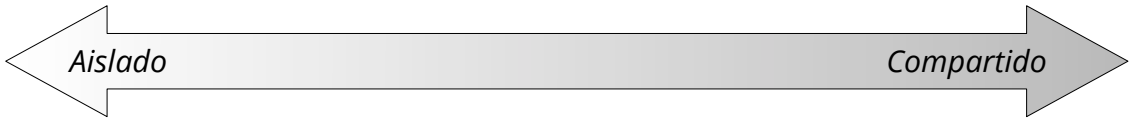


Figura 31. Separación continua del aislamiento de datos.

Además de seguridad y la privacidad, la personalización del modelo de datos base del sistema MT es fundamental; los tenants deben de ser capaces adaptar el modelo de datos a sus necesidades [112]. Según [23], cuanto más aislado sea el enfoque más fácil será de personalizar pero también más caro es el hardware y el mantenimiento. No existen pruebas que permitan afirmar que el lado aislado es mejor que el compartido y viceversa y elegir un grado de aislamiento en nuestro sistema depende de muchos factores que serán analizados posteriormente.

El artículo [15] describe tres aproximaciones para la implementación de bases de datos multi-tenant. Por su parte, [103] considera también tres enfoques semejantes pero usa nomenclatura diferente (Figura 32). En este trabajo hemos unificado las vocabularios para expresar propuestas análogas:

- ***Bases de datos separadas*** *(Separate databases* [103] *o Shared Machine* [15]*):* La información de los tenants es almacenada en bases de datos diferentes. Alto grado de aislamiento.

- ***Base de datos compartida, tablas separadas*** *(Shared database, separate schemas* [103] o *Shared Process* [15]): Los tenants comparten la base de datos pero la información se almacena en tablas [15] o esquemas (conjunto de tablas) diferentes [103]. Grado medio de aislamiento.

- ***Tablas compartidas*** (*Shared table* [15] o *Shared database, shared schemas* [103]*)*: Los tenants comparten tanto la base de datos como las tablas. En este caso cada registro de información debe incorporar el tenant al que pertenece. Nivel más bajo de aislamiento.

Figura 32. Aproximaciones de implementación MT y zona del espectro donde situarlas

Algunos autores consideran *multi-tenancy puro* [91] cuando se maximiza el uso compartido de recursos y se implementan enfoques con niveles bajos de aislamiento (tablas compartidas). Cualquier variación que no considere esta optimización está fuera del concepto y la consideran como *semi multi-tenant*.

Pasamos a analizar en detalle cada una de los enfoques anteriores, estudiando sus ventajas y desventajas [103].

5.1.1 Bases de datos separadas

En este caso, los datos de cada propietario se almacenan en bases de datos diferentes. Se trata de la opción más aislada para la capa de datos.

El código de la aplicación y los recursos hardware son normalmente compartidos por los diferentes tenants, pero cada uno de ellos dispone de su propio conjunto de datos que se encuentra aislado de los datos del resto.

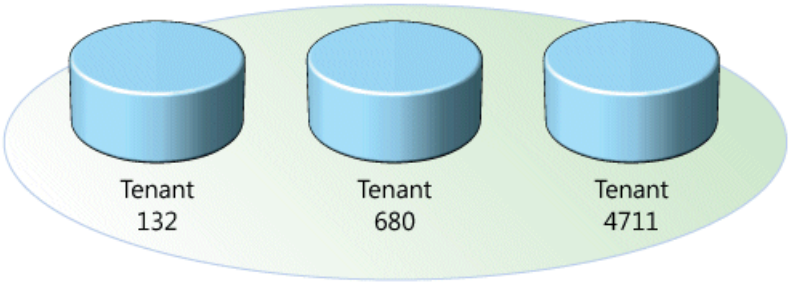


Figura 33. Bases de datos separadas. Cada tenant tiene su propia base de datos

El nivel administrativo debe asociar correctamente cada base de datos a los propietarios. La propia seguridad del sistema de gestión de base de datos previene el acceso no autorizado accidental o malicioso a los datos de otro tenant.

Ventajas

- Facilidad en la personalización de la capa de datos para adaptarse a las necesidades de cada cliente.
- Seguridad en el acceso a datos.
- Fácil recuperación y restauración del sistema en caso de fallo, ya que se limitaría a volcar de nuevo la copia de seguridad individual.
- Migración de datos de cliente más sencilla.
- Mayor tolerancia a fallos debido a la separación de la capa de datos. Para que se caiga el sistema al completo, deben de fallar todas las bases de datos.
- Mayor facilidad en la programación del sistema. Al estar cada cliente en una base de datos diferente, las consultas se simplifican, ya que no es necesario especificar filtros de propietario.

Desventajas

- Alto coste del mantenimiento de los equipos y en el sistema de *backup* y recuperación de datos
- Mayor coste de la infraestructura hardware debido a que el número de tenants que el sistema puede soportar depende del número de bases de datos que pueda hospedar el servidor.

En **conclusión,** podemos decir que la separación de bases de datos es la aproximación "*premium*", ya que es apropiada para aquellos tenants que prefieren pagar más dinero

por obtener mayores prestaciones como la personalización y seguridad del sistema. Ejemplo de ello son los clientes del sector médico o bancario que por regla general, sus aplicaciones presentan un gran aislamiento de la capa de datos y no consideran una solución que no presente esta separación.

En [103], los autores denominan este enfoque como *Shared Machine*. Cada tenant tiene una base de datos diferente, estando en una máquina compartida. Es importante mencionar la imposibilidad en este enfoque de realizar *operaciones administrativas masivas*, de forma que cada base de datos ejecutaría sus propias consultas. Por el contrario, nos indica igualmente la facilidad en la migración de los datos.

5.1.2 Base de datos compartida, tablas separadas

En esta aproximación, múltiples tenants son alojados en la misma base de datos, pero las tablas de cada uno están agrupadas en esquemas diferentes, uno por cada tenant (Figura 34).

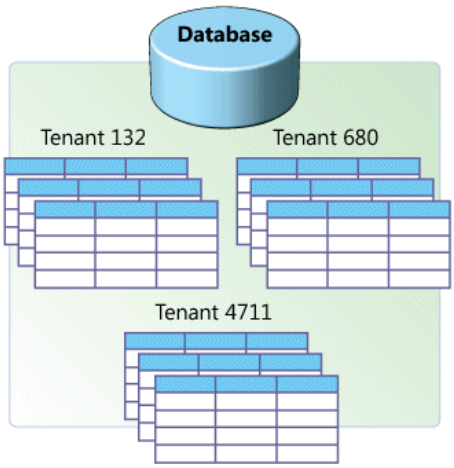


Figura 34. Base de datos compartida, tablas separadas. Cada tenant tiene sus propias tablas.

Cuando un tenant se conecta por primera vez al servicio, el sistema creará el conjunto de tablas y las asociará a su cuenta como propietaria. Al igual que en la aproximación aislada, la separación de tablas facilita la implementación al disminuir la complejidad de las consultas. Del mismo modo, también se ve beneficiada la personalización ya que, aunque las tablas son clonadas de un conjunto inicial común, estas pueden modificarse en función de las necesidades del cliente.

Si el sistema de gestión de bases de datos (SGBD) permite la definición de esquemas, esta aproximación presenta las mismas ventajas en cuanto a seguridad que la separación de bases de datos, ya que, a nivel lógico no podemos considerar diferencia. En caso de que no los soporte, puede que se precise un poco más de lógica en la aplicación pero igualmente podremos beneficiarnos de las ventajas. Por tanto, es una buena elección ya que elimina la restricción de escalabilidad por el número de bases de datos que pueda soportar nuestro servidor.

Ventajas:

- Implementación sencilla de las consultas de la aplicación. Sobre todo si hay soporte a esquemas pero si no, también es bastante fácil ya que basta con sustituir el nombre de la tabla.
- Personalización. Cada tabla puede ser modificada de forma independiente.
- Seguridad en acceso de la capa de datos, al estar implementada a nivel de SGBD.
- Fácil recuperación, restauración y migración de datos.
- Disminución del coste de infraestructura hardware.

Desventajas:

- Aumento del coste de restauración de datos. El fallo de los datos de un propietario implica la restauración de la base de datos completa, con todos los esquemas. Los propietarios, aún sin perdida en sus datos, se verían afectados por esta restauración.

La separación de esquemas es apropiada para las aplicaciones que requieren un número bajo de tablas, del orden de 100 tablas por tenant o menos. Esta aproximación permite alojar un mayor número de propietarios por servidor, por lo que puede ser ofrecida a un menor coste. Sin embargo, es necesario informar a los usuarios de que sus datos tendrán una misma localización, cuestión que puede minar la contratación del sistema.

Según [103], al haber una sola base de datos los tenants pueden compartir pools de conexiones a la misma. En este sentido hay una conocida desventaja y es que todas las conexiones están ligadas a una principal que tiene acceso a todo. Por tanto, la seguridad tiene que estar implementada a nivel de aplicación y no de SGBD. Las operaciones administrativas masivas pueden ejecutarse, aunque pueden ser problemáticas en algunas

bases de datos. Por ejemplo, la modificación del esquema base implica la ejecución de sentencias DDL y presenta problemas de rendimiento si el sistema se encuentra en ejecución.

5.1.3 Tablas compartidas

En este diseño se comparten la base de datos y el esquema para los clientes. Por tanto, los propietarios comparten las mismas tablas y un atributo *Tenant_ID* asocia cada registro a su propietario correspondiente. En modelos relacionales, el campo Tenant_ID presente en las tablas de los tenants es una clave externa a la tabla de administración de tenants (Figura 35 y Figura 36).

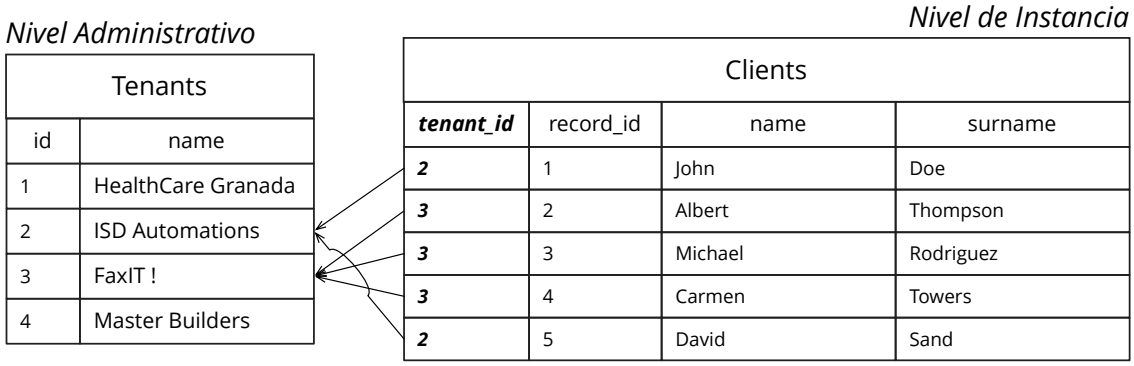


Figura 35. Tablas compartidas: El campo Tenant_ID determina el propietario del registro

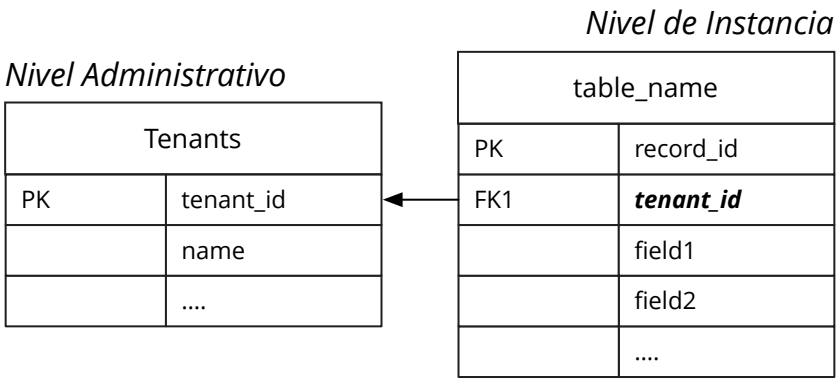


Figura 36. Tabla administrativa de gestión de tenants y relación con tablas del nivel de instancia.

De las tres aproximaciones vistas anteriormente, el de *tablas compartidos* presenta el menor coste en cuanto a hardware y copias de seguridad, ya que permite alojar el mayor número de tenants en la base de datos. Sin embargo, el desarrollo de las aplicaciones es más complicado, ya que es necesario controlar la seguridad en el acceso

a los datos. La privacidad es compleja de implementar a nivel de SGBD y es necesario que el programador sea el que asegure que cada tenant acceda única y exclusivamente a sus datos.

Otra desventaja que tiene esta aproximación, al igual que separación de esquemas es que el proceso de restauración de datos incluye a todas las tenancies, independientemente de que sus datos hayan sido los afectados. Además, si el número de filas en las tablas es muy notable, el tiempo de respuesta será muy elevado para todos los tenants, cuando puede que alguno de los vecinos solamente tenga unos pocos registros.

Ventajas:

- Coste mínimo de infraestructura y mantenimiento. Reducción máxima de precios para el cliente.
- Optimización de recursos. Número elevado de tenants.

Desventajas:

- El fallo de los datos de un propietario implica la restauración de la base de datos completa, afectando a usuarios sin problemas.
- La migración de un cliente implica un filtro y selección de tablas para la extracción de datos de cliente. Esta complejidad es mayor para modelos relacionales.
- Personalización compleja, a través de metadatos de extensión y técnicas de mapeo.
- Implementación compleja de las consultas de la aplicación. Acceso lento en el caso de muchos registros.
- La seguridad debe de estar implementada a nivel de aplicación.
- Fácil recuperación, restauración y migración de datos.

Según [15], la seguridad sólo puede ser implementada a nivel de SGBD sólo si el sistema permite la definición de privilegios a nivel de registros. Señala los inconvenientes ya mostrados y como ventaja destaca que las operaciones administrativas masivas pueden ser articuladas de manera sencilla.

Tablas compartidas es apropiado en las aplicaciones donde el número de registros no sea muy elevado y permite alojar el mayor número de tenants con el menor número de

servidores. El cliente en este caso está dispuesto a sacrificar aspectos importantes como la seguridad y el rendimiento, a cambio de menores precios.

5.2 Técnicas de mapeo de esquemas

Los modelos aislados permiten implementar multi-tenancy de forma sencilla pero hemos visto que presentan inconvenientes. Entre otros, la escalabilidad del sistema se ve limitada por el número de tablas que éste pueda alojar, tanto a nivel de espacio como por rendimiento de memoria [113].

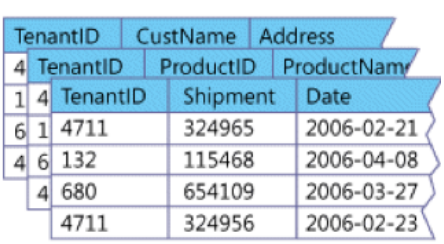
La forma más sencilla es usar un enfoque altamente compartido en el que los tenants usan las mismas tablas reduciendo su número. Sin embargo, esta decisión afecta a la personalización de la aplicación, ya que la modificación de la estructura común afectaría a todos los suscriptores. La solución consiste en aplicar técnicas de mapeo de esquemas que permitan a los tenants adaptar las aplicaciones a sus necesidades.

Estas técnicas permiten a cada suscriptor tener un esquema lógico personalizado que es mapeado a las tablas físicas comunes en un entorno compartido.

A continuación hacemos un repaso del estado del arte para analizar los enfoques de mapeo más destacados.

5.2.1 Esquema Básico

Basic Layout [57] es el modelo más básico para implementar multi-tenancy. Tal y como vemos en la Figura 37 [15], [103]. consiste en mantener tablas comunes para todos los tenants, incluyendo en cada registro un campo Tenant_ID de enlace con la tabla administrativa de tenants.



TenantId	Account	Name	…	Val0	…	Val100
1041	0021	Acme		1/3/95		----
1041	0029	Ball		3/7/72		----
1053	0016	Gump		red		----
1053	0049	Wonk		blue		----

Figura 37. Ejemplos de tablas con basic layout (tablas compartidas)

En realidad este modelo se corresponde el enfoque de *Tablas Compartidas* del modelo aislado, sin posibilidad de personalización. Lo incluimos dentro de los patrones por considerar un patrón de personalización nulo.

5.2.2 Tablas Privadas

Private Tables [57], [113] se corresponde con el modelo base del mismo nombre. En este caso el modelo de extensión es intrínseco. La extensibilidad de la aplicación viene dada por la privacidad en las tablas. Cada tenant almacena sus datos dentro de la misma base de datos, pero en tablas diferentes.

En la arquitectura del sistema, el componente de procesamiento de consultas debe únicamente renombrar las tablas de tenant. En la siguiente Figura 38 [57] podemos ver el ejemplo de la tabla de clientes para tres tenants diferentes. El esquema base común es sencillo, incluye el identificador del cliente y el nombre. Sin embargo cada cliente ha modificado las tablas para añadir los campos necesarios; el Tenant 35 no lo ha modificado, Tenant 17 ha añadido "hospital" y "camas" mientras que el Tenant 42 "*Dealers*" (compradores).

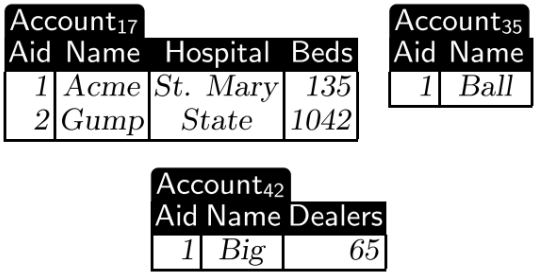


Figura 38. Ejemplo de extensión de Private Table Layout

[103] denomina a este enfoque *Columnas personalizadas* (Custom Columns) y lo considera el enfoque más sencillo que permite la extensibilidad del modelo de datos ya que las columnas pueden ser añadidas por los tenants directamente (Figura 39).

EmployeeID	FirstName	BirthDate	LastReview	Branch	401k
653	Pat	1952-11-04	null	"San Francisco"	true
1310	Tom	1949-12-14	2006-01-30	"London"	null
280	Surendra	1973-09-12	2005-11-08	"Bangalore"	null
985	Christine	1981-03-26	2006-06-09	"San Francisco"	false
1701	Gordon	1964-08-20	null	"Toronto"	null

Figura 39. Adicción directa de dos nuevos campos en la tabla empleados.

Lógicamente esta aproximación es válida para un enfoque aislado de la capa de datos. Quizá sea el método más sencillo ya que no implica mucho trabajo extra en la transformación de consultas, al estar toda la información dentro de la misma tabla.

Sin embargo, desde el punto de vista arquitectónico, supone un esfuerzo extra para la implementación, ya que permite a los tenants modificar la estructura de las tablas.

5.2.3 Tablas de Extensión

Las dos aproximaciones anteriores se pueden combinar separando los campos personalizados en tablas de extensión separadas [57] (Extension Tables). De este modo, mantenemos el enfoque compartido para un esquema básico de cada tenant (tablas base), y sacamos fuera los campos personalizados de cada uno de ellos (tablas de extensión). Según [57], una referencia a la fila de la tabla base debe ser añadida a la tabla de extensión (Figura 40). [23] considera un enfoque análogo (que denomina al *Custom Extensions*) y podemos ver un ejemplo del mismo en la Figura 41.

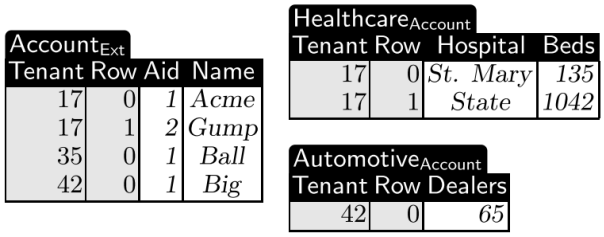


Figura 40. Tablas de extensión.

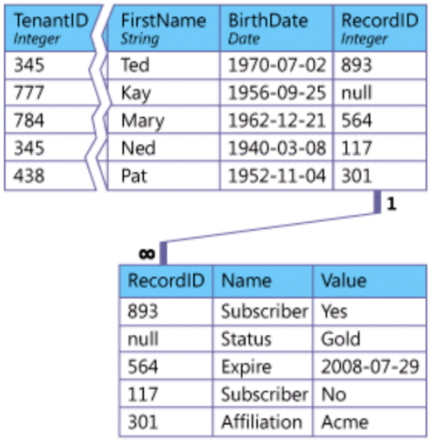


Figura 41. Custom Extensions.

Este enfoque nos proporciona un mayor soporte a la extensibilidad de la aplicación en detrimento del rendimiento. Esto es debido a que *joins* adicionales han de ser procesados por la capa de abstracción de consultas para la recuperación de datos.

5.2.4 Tabla Universal

Una *Tabla Universal* (Universal Table) [57] es una estructura genérica con un atributo *Tenant_ID*, otro identificador de la tabla y un número grande de atributos para alojar datos genéricos (Figura 42).

Universal

Tenant	Table	Col1	Col2	Col3	Col4	Col5	Col6
17	0	1	Acme	St. Mary	135	–	–
17	0	2	Gump	State	1042	–	–
35	1	1	Ball	–	–	–	–
42	2	1	Big	65	–	–	–

Figura 42. Tabla Universal

[23] explica un enfoque análogo con el nombre de *Fixed Extension Set* (Figura 43) y en él las columnas de datos deben tener un tipo de dato flexible, que pueda ser convertido mediante operaciones de CASTING a otros, como es el caso de VARCHAR.

La enésima columna de la tabla lógica fuente de cada tenant, es mapeada en la enésima columna de la Tabla Universal. De este modo, los suscriptores pueden extender su conjunto de datos según sus necesidades. Este enfoque es el adoptado por Salesforce.com [114]

TenantID *Integer*	FirstName *String*	BirthDate *Date*	Custom1 *Integer*	Custom2 *String*	Custom3 *Untyped*
345	Ted	1970-07-02	null	Paid	null
777	Kay	1956-09-25	23	null	null
784	Mary	1962-12-21	null	null	null
345	Ned	1940-03-08	null	Paid	null
438	Pat	1952-11-04	null	San Francisco	Yes

Figura 43. Fixed Extension Set

Tener las columnas de extensión en la misma tabla mejora el rendimiento de la aplicación al no tener la capa de abstracción de consultas que realizar operaciones extra. Sin embargo, este enfoque presenta la limitación del número de campos de extensión. La capacidad de extensión de la aplicación vendrá dada por el número de campos extra añadidos en tiempo de diseño, lo que puede limitar la personalización.

Al escoger este enfoque, el número de filas será muy elevado, incluso cuando tengamos tablas fuente con pocos registros. Además, el SGBD deberá tratar con muchos valores nulos, correspondientes a los campos de extensión no utilizados. Este tratamiento no suele estar optimizado en la bases de datos relacionales comerciales.

5.2.5 Columnas Dispersas

En [113] se propone *Sparse Columns* como un enfoque utilizando esta tecnología en la que se mezcla la Tabla Universal con las *Tablas de Extensión*. Los campos de extensión de cada suscriptor se anexan a la tabla base convencional. Podemos ver un ejemplo en la Figura 44.

Account						
Tenant	Aid	Name	*SPARSE*			
17	1	*Acme*	Hospital	*St. Mary*	Bed	*135*
17	2	*Gump*	Hospital	*State*	Bed	*1042*
35	1	*Ball*				
42	1	*Big*	Dealer	*65*		

Figura 44. Enfoque Sparse Columns

La aplicación debe asegurar que cada tenant solamente use aquellas columnas que ha declarado; tanto en las lecturas como en las escrituras. Es por ello por lo que se debe llevar un registro en metadatos de las columnas definidas por los tenants, su propiedad, tipo y descripción.

Este enfoque reduce el número de tablas necesarias con respecto a las tablas privadas y las tablas de extensión y por ende incluye positivamente en el rendimiento. Por el contrario el rendimiento se ve afectado al tener que tratar con columnas dispersas. Sparse Columns está implementada en Microsoft SQL Server 2008 de forma que permite optimizar el espacio para valores NULOS dentro de la base de datos [115].

5.2.6 Campos preasignados

Campos pre-asignados (*Preallocated fields*) [103] redefine el enfoque *Sparse Columns*. Nos propone una forma de extender el modelo de datos que consiste simplemente en dejar una serie de campos en blanco dentro de cada tabla y darle a cada tenant la posibilidad de usarlos "a su antojo". Como vemos en la Figura 45, disponemos de tres campos que cada tenant puede usar dependiendo de sus necesidades.

TenantID	FirstName	BirthDate	C1	C2	C3
345	Ted	1970-07-02	null	"Paid"	null
777	Kay	1956-09-25	"66046"	null	null
1017	Mary	1962-12-21	null	null	null
345	Ned	1940-03-08	null	"Paid"	null
438	Pat	1952-11-04	null	"San Francisco"	"Yes"

Figura 45. Tabla con campos preallocated desde C1 hasta C3

La pregunta es la siguiente ¿Qué ocurre con los tipos de datos? Podríamos pensar en asignar en diseño un tipo de dato fijo a cada campo, pero supondría una restricción innecesaria para los tenants ya que se verían limitado los datos en función del tipo que el arquitecto hubiera asignado al campo. Por ejemplo, se puede dar el caso que el tenant necesitara 3 campos de tipo *String* en su personalización, pero no pudiera porque existe un tipo *Bool* y otro *Integer* que le impiden almacenar este tipo de datos. La solución a este problema consiste en almacenar los tipos en una tabla separada de metadatos. Por ejemplo, en la Figura 46 [103] vemos representado como un formulario web de un determinado tenant hace uso de este enfoque para almacenar el Código Postal

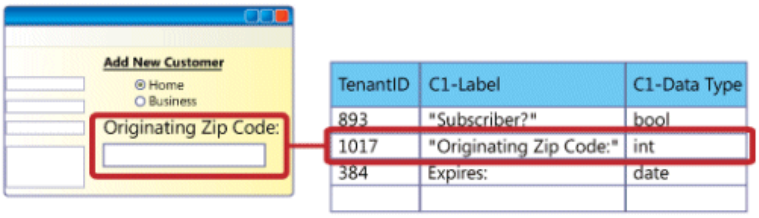


Figura 46. Almacenamiento de los tipos en una tabla separada de metadatos

El diseño de la tabla de metadatos puede hacerse usando una misma tabla para todos los campos pre-asignados, o tablas separadas para cada campo. Vemos las una comparativa de las posibilidades en la Figura 47 [103].

TableID	C1-Lable	C1-DataType	C2-Lable	C2-DataType
893	"Subscriber?"	bool	"Subscription Code"	string
1017	"Originating Zip Code:"	int	null	null
564	"Expires"	date	"Auto Review?"	bool

Table: *C1ExtensionTable*

TableID	C1-Lable	C1-DataType
893	"Subscriber?"	bool
1017	"Originating Zip Code:"	int
564	"Expires"	date

Table: *C2ExtensionTable*

TableID	C2-Lable	C2-DataType
893	"Subscription Code"	string
564	"Auto Review?"	bool

Figura 47. Posibilidades de diseño. Una tabla única arriba y tablas separadas abajo.

La ventaja de usar tablas separadas es el ahorro de almacenamiento, solamente tendrán entrada aquellos tenants que hagan uso del campo en concreto. En el modelo de tabla única existirá una entrada para todos los tenants que al menos usen uno de los campos preasignados. De esta forma la existencia de campos nulos será común en aquellos tenants que no hagan uso del campo. Por el contrario, este enfoque supone una disminución del rendimiento debido al aumento de complejidad por los JOIN necesarios en la obtención de los campos preasignados de cada tenant.

5.2.7 Pares Nombre-Valor

El enfoque campos pre-asignados de la anterior sección es una manera sencilla de articular la extensibilidad de la aplicación. Sin embargo, este modelo tiene limitaciones; debemos decidir en tiempo diseño cuántos campos personalizados tendrá cada tabla. Pocos campos disminuyen la capacidad de personalización de los tenants mientras que muchos pueden llegara convertir la base de datos en un derroche disperso y lento.

El enfoque de Pares Nombre-Valor (*Name-Value Pairs*) [103] propone una solución a estas limitaciones permitiendo a cada usuario definir sus propios campos.

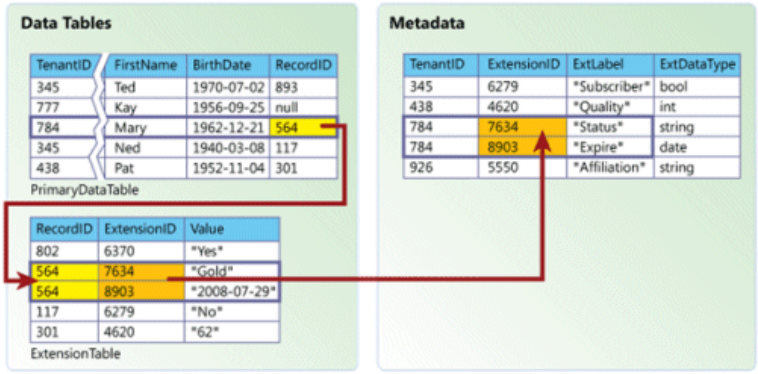


Figura 48. Tabla de extensión con enfoque par Nombre-Valor

Tal y como vemos en la Figura 48 [103], en este método el sistema debe proporcionar un identificador único para cada registro en la tabla base. Este identificador lo relaciona con la tabla de extensión conteniendo los valores de los atributos personalizados de cliente en una tabla de metadatos.

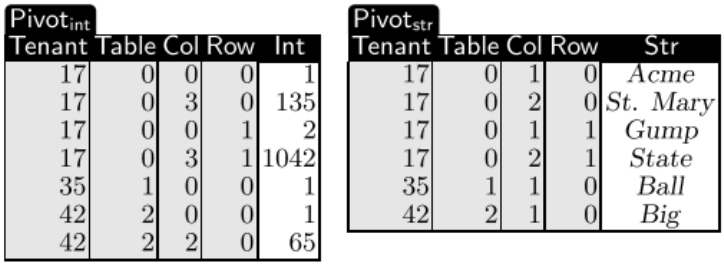
Este enfoque permite la extensibilidad del modelo de datos manteniendo los beneficios de coste de la aproximación compartida. La principal desventaja es el coste computacional de relacionar las tablas de extensión con los registros de las tablas base. Podemos anticipar que este enfoque es una de las mejores opciones si la aplicación va a ser compartida en la capa de datos permitiendo una fuerte personalización a nuestros clientes.

5.2.8 Tabla Pivotante

Una tabla pivotante (*Pivot Table Layout*) [57] es una estructura genérica donde cada campo de tabla lógica del tenant es trasladado a una fila dentro de la *Pivot Table*. Además de las columnas *Tenant*, *Table* y *Row* que forman parte de la definición de la estructura *Pivot Table*, existe una columna *Col* que identifica qué columna de la tabla lógica representa esta entrada dentro de la tabla pivotante.

La columna de datos la podríamos unificar en un tipo flexible como VARCHAR, que lo pudiéramos convertir a otros tipos de datos. En este caso nos encontraríamos con que la Pivot Table se convierte en una *Universal Table* para el modelo de almacenamiento DSM (*Decomposition Storage Model* [116]).

El enfoque se puede mejorara si creamos una tabla pivotante por cada tipo de datos (Figura 49 [57]). Incluso, para soportar el indexado podemos crear dos tablas por tipo, una con índices y otra sin él. Cada valor lo ubicaremos en la tabla pivotante que le corresponda dependiendo si necesita ser indexado o no.



Pivot$_{int}$

Tenant	Table	Col	Row	Int
17	0	0	0	1
17	0	3	0	135
17	0	0	1	2
17	0	3	1	1042
35	1	0	0	1
42	2	0	0	1
42	2	2	0	65

Pivot$_{str}$

Tenant	Table	Col	Row	Str
17	0	1	0	*Acme*
17	0	2	0	*St. Mary*
17	0	1	1	*Gump*
17	0	2	1	*State*
35	1	1	0	*Ball*
42	2	1	0	*Big*

Figura 49. Pivot Table Layout optimizada con división de pivot tables según tipo de dato

Esta aproximación elimina el tratamiento masivo de valores nulos, sin embargo mantiene más metadatos que datos (en gris en la figura). La capa de abstracción de consultas debe realizar una gran cantidad de *joins* para recuperar la tabla lógica, lo cual incurre en sobrecarga para el sistema.

En [113] podemos encontrar un experimento de comparación de rendimiento con este enfoque. En [18] se recoge un estudio de comparación entre las tablas pivotantes (llamadas verticales) y las tablas horizontales convencionales. El estudio concluye a favor de las primeras debido a que optimizan mejor el acceso selectivo.

5.2.9 **Tabla de pedazos**

Chunk Table Layout [57] propone una nueva estructura genérica denominada *Chunk Table*. "*Chunk*" significa *pedazo, parte, fragmento o trozo* en inglés. Este método es particularmente efectivo cuando los datos pueden ser segmentados en subconjuntos densos y bien definidos. Como podemos ver en la Figura 50 [57] se trata de una *Pivot Table* con la diferencia que existe más de un campo lógico mapeado en la tabla. En el ejemplo observamos cómo son dos los campos incluidos en el pedazo.

$Chunk_{int|str}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	1	*Acme*
17	0	1	0	135	*St. Mary*
17	0	0	1	2	*Gump*
17	0	1	1	1042	*State*
35	1	0	0	1	*Ball*
42	2	0	0	1	*Big*
42	2	1	0	65	–

Figura 50. Chunk Table Layout

El campo "Col" de la tabla dinámica ha sido sustituido por el campo "Chunk" que identifica el pedazo de la tabla lógica al que corresponde la entrada. Este enfoque elimina por tanto la cantidad de metadatos almacenados y mejora el rendimiento al disminuir el número de *joins* a ejecutar.

5.2.10 Desdoblamiento en pedazos

Esta técnica (*Chunk Folding* [57]) consiste en dividir las tablas lógicas en diferentes *Tablas de Pedazos* que combinamos posteriormente para extraer las tablas lógicas. El enfoque trata de optimizar la carga al máximo de metadatos repartiéndolos entre tablas convencionales y un conjunto fijo de Tablas de Pedazos. De esta forma, la carga de metadatos se reparte entre las tablas base y el conjunto de tablas de extensión.

Una posible opción en combinar el modelo de Tablas de Extensión con Chunk Table. Esto es, separamos la información de extensión en tablas de en una única Chunk Table que registra las personalizaciones de los clientes. Podemos ver el ejemplo en la Figura 51 [57].

$Account_{Row}$

Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

$Chunk_{Row}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	135	*St. Mary*
17	0	0	1	1042	*State*
42	2	0	0	65	–

Figura 51. Chunk Folding

Un buen rendimiento se consigue mapeando los datos más utilizados en las tablas convencionales y el resto en las de extensión. De este modo disminuimos estadísticamente el número de operaciones *join*.

5.2.11 Lenguaje de marcas

Consiste en añadir una columna a la tabla base conteniendo un campo semiestructurado (XML, JSON, etc.) que describa las extensiones de cada tenancy. Un ejemplo de este enfoque es *XML Layout* [113], como vemos en la Figura 52, existe un campo *Ext_XML* que al procesarlo nos permite obtener los valores de extensión de cada cliente.

Account

Tenant	Aid	Name	Ext_XML
17	1	Acme	<ext><hospital>St. Mary</hospital> <beds>135</beds></ext>
17	2	Gump	<ext><hospital>State</hospital> <beds>1042</beds></ext>
35	1	Ball	
42	1	Big	<ext><dealers>65</dealers></ext>

Figura 52. XML Layout

IBM implementa este enfoque con *pureXML* [117], [118]. Esta versión comercial ofrece de forma nativa funciones de consulta híbrida que nos permiten acceder tanto a la información estructurada como semiestructurada. La Figura 53 nos muestra un ejemplo de consulta con soporte XML Nativo en pureXML.

```
SELECT b.Tenant, b.Aid, b.Name,
       e.Dealers
FROM   Accounts b,
       XMLTABLE('i/ext' PASSING b.Ext_XML AS "i"
         COLUMNS
           Dealers INTEGER PATH 'dealers'
       ) AS e
WHERE Tenant = 42 AND Aid = 1;
```

(a) Physical SELECT Query

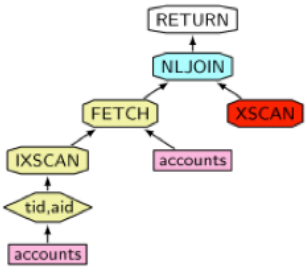


Figura 53. Ejemplo de consulta pureXML

Este es un aspecto primordial para el uso de este enfoque, ya que, en caso contrario disminuiría el rendimiento de la aplicación de manera considerable al tener que procesar nosotros mismos el XML, implicando también un coste mayor de desarrollo. A favor, nos encontramos con una gran configurabilidad por parte del tenant al no tener límites como en el caso de la Tabla Universal o Columnas Dispersas.

En [119] los autores proponen una combinación de este modelo con las Tablas de Extensión almacenando datos estructurados en tablas separadas que contengan los campos personalizados de los clientes.

5.2.12 HBase

HBase (Apache) es la versión de código abierto de *Google BigTable* [42]. En HBase, las columnas son agrupadas por DDL en *familias de columnas*. Dentro de una familia se pueden crear más columnas de forma dinámica y diferentes filas de una tabla pueden usar la misma familia de columnas de forma distinta. Los datos de una misma familia se almacenan juntos en disco y memoria por lo que una familia es básicamente una Tabla Pivotante.

HBase es una base de datos no relacional que permite una enorme escalabilidad. Ofrece a su vez muchas más prestaciones y es por ello por lo que ha sido elegida como SGBD de algunas de los proyectos más ambiciosos a nivel mundial como *Facebook Messages* [121].

5.2.13 Tablas elásticas de extensión

EET (*Elastic Extension Tables* [122]) propone un enfoque en el que la información personalizada se articula mediante tablas de extensión. En esta propuesta la información se divide en CTTs *(Common Tenants Tables*) y VETs (*Virtual Extension Table*) (Figura 54 [122]).

Los tenants pueden mapear el esquema de la base datos en las VETs incluyendo incluye de nuevos campos y tablas. Por tanto, ETT es un enfoque que añade elasticidad al modelo de Tablas de Extensión al poder modificar dinámicamente el modelo personalizado de datos de los clientes.

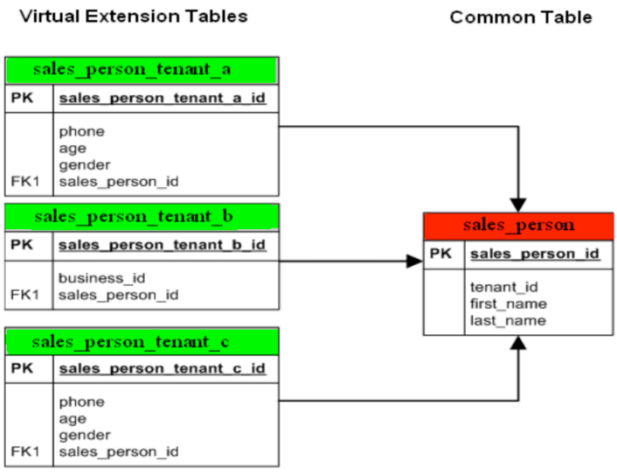


Figura 54. Enfoque EET

5.3 La elección de un enfoque

Cada uno de los enfoques descritos anteriormente ofrece su propio conjunto de ventajas y desventajas que hacen que sea un modelo apropiado a seguir en algunos casos y en otros no, dependiendo de aspectos técnicos y de negocio.

En [123] los autores llevan a cabo un estudio para la evaluación del rendimiento; los mejores resultados los obtienen aquellos enfoques situados más cerca del extremo aislado del espectro. Sin embargo, dado que cada tenant tiene sus propios recursos estos pueden estar desaprovechados; lo cual afecta a la eficiencia del sistema, aumenta costes y limita el número de tenants posibles a alojar. En [103] se analizan factores clave y reflexiona sobre ellos a la hora de seleccionar un modelo y su grado de aislamiento.

5.3.1 Factores Económicos

Las aplicaciones optimizadas para el enfoque compartido tienden a tener un mayor coste inicial por el esfuerzo extra en el desarrollo. Este coste inicial, sin embargo se ve compensado con una mejor amortización al tener menor coste operacional.

El enfoque compartido tiende a tener un mayor tiempo de desarrollo debido a la complejidad, frente a las aplicaciones con la aproximación aislada, en la que las que el tiempo de programación debe ser menor.

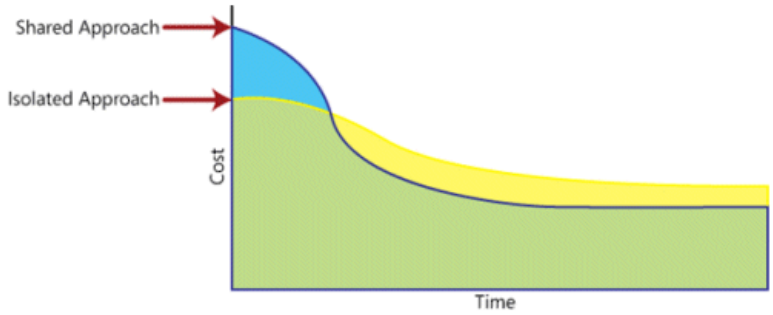


Figura 55. Comparación de coste sobre tiempo para dos tipos de aplicaciones

Tal y como apreciamos en la Figura 55 [103], el enfoque compartido reporta mayores beneficios a largo plazo, pero tiene un alto coste inicial. En caso de que no se pueda realizar una inversión fuerte en el comienzo del proyecto, debemos entonces optar por la opción aislada.

5.3.2 Seguridad

Dado que la aplicación almacenará información de especial relevancia para los usuarios, estos esperarán un alto nivel de seguridad en nuestro sistema. No debe darse en ningún caso la situación en que un tenant pueda acceder a la información de otro, por error o por malas intenciones.

Un error típico es considerar que el enfoque aislado es la única posibilidad que asegura un adecuado nivel de privacidad en la capa de datos. Sin embargo, un enfoque compartido puede proporcionar las mismas garantías, tan sólo que es necesaria una mayor complejidad en el diseño.

5.3.3 Tenants

El número, naturaleza y necesidades de los propietarios que esperamos dar servicio en la aplicación afecta la arquitectura de la capa de datos de muy diferentes formas. Debemos reflexionar sobre las siguientes preguntas que nos inclinarán hacia un enfoque u otro. La Figura 56 [103] muestra ilustra el modelo es más adecuando en función de los aspectos planteados.

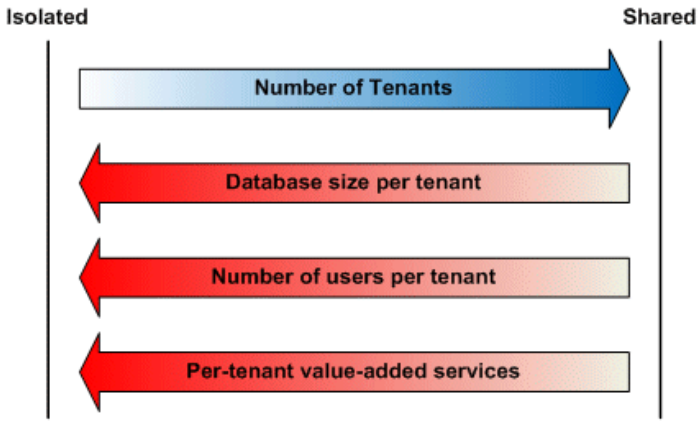


Figura 56. Elección de un enfoque en función de las características de los tenants

- ¿**Cuántos propietarios** esperamos tener? Quizá no podamos estimar el número de manera objetiva, pero si podemos pensar en tamaños de mayor nivel de abstracción: ¿Estamos desarrollando la aplicación para cientos? ¿Para miles? ¿Cientos de miles? Cuanto mayor sea el número estimado, más debemos tender hacia una opción compartida.
- ¿Cuánto **espacio** necesitará de media cada propietario? Si pensamos que cada propietario almacenará una gran cantidad de datos, quizá sea mejor una separación de las bases de datos. Efectivamente, grandes requerimientos de espacio pueden forzarnos a separar posteriormente los datos de cliente en bases de datos diferentes. Por esto es mejor tomar la decisión en la fase de diseño.
- ¿Cuántos **usuarios concurrentes** esperamos tener en el sistema? El número de conexiones se puede ver comprometido en el enfoque compartido, por tanto un aislamiento de datos mejorará las prestaciones al cliente.
- ¿Ofrecemos **servicios adicionales** de copias de seguridad y restauración? En este caso, tal y como hemos visto anteriormente, el enfoque aislado favorece estas funcionalidades.

5.3.4 Marco Legal

Compañías, organizaciones y Gobiernos, están sujetos a leyes que afectan a la seguridad de los datos y características del almacenamiento. Por tanto, es muy aconsejable realizar una investigación del marco legal en que se encuadrarán nuestros clientes potenciales para ajustar estas regulaciones en nuestro sistema.

5.3.5 Habilidades para el desarrollo

El diseño de aplicaciones MT requiere unas habilidades poco comunes debido a la novedad de esta arquitectura. Es muy difícil encontrar expertos en esta materia. Si nuestros arquitectos software y desarrolladores no tienen experiencia en SaaS, o bien contratamos personal que la tengan o bien formamos a nuestro equipo. En ciertos casos, con poca experiencia en multi-tenancy, puede ser una mejor opción el enfoque aislado, debido a que el desarrollo se asemeja más al de las aplicaciones tradicionales.

5.4 Escalabilidad en la capa de datos

La escalabilidad se refiere a la capacidad que tiene una aplicación de incorporar más datos en la aplicación o un mayor número de usuarios. Esto es sin duda más importante aún en SaaS debido a su entorno compartido ya que el número de usuarios finales se multiplica por el número de clientes/empresas suscritas a la aplicación.

Nuevamente, la escalabilidad es un factor que se complica en las aplicaciones SaaS con respecto a las aplicaciones tradicionales. Debido al carácter multi-tenant (con el manejo de información de múltiples clientes) **el volumen de datos crece mucho más rápido que una aplicación tradicional**.

El paso de single a multi-tenant en la arquitectura, se puede comparar como un ascenso en la liga. Las reglas del juego son las mismas, pero la repercusión y el número de usuarios es muchísimo mayor. Debemos tener en cuenta que estamos construyendo una aplicación a escala de Internet con miles de millones de usuarios potenciales.

Una base de datos compartida deberá ser escalada cuando:

- No cumpla con las métricas mínimas de rendimiento.
- Existan demasiadas conexiones concurrentes o el tamaño de la base de datos implica que las operaciones tardan demasiado en ejecutarse.

Las bases de datos pueden ser *escaladas hacia arriba* (*scale-up*) cambiando el servidor por otro más potente, mayor memoria y rapidez de unidades de disco y *escaladas hacia fuera* (*scale-out*) distribuyendo la base de datos en diferentes servidores.

Existen dos técnicas principales para hacer el scale-out de una base de datos: *replicación y particionamiento.* La replicación consiste en hacer una copia total o parcial de la base de datos en otras localizaciones y mantener la copias sincronizada s con la original. Dentro de ella, la replicación *single-master* es más sencilla de gestionar, ya que sólo la base de datos original puede ser modificada. Por el contrario, en la replicación *multi-master* todas las copias pueden ser modificadas por lo que tiene que existir algún mecanismo de reconciliación de cambios entre las distintas versiones de los datos.

El particionamiento consiste en "podar" subconjuntos de los datos y realojarlos en otra base de datos o en tablas diferentes dentro de la misma base de datos. Esta división puede ser *horizontal* o *vertical*. En el particionamiento horizontal, la base de datos se divide en dos o más bases de datos, con el mismo esquema y estructura, pero con menos filas. El particionamiento vertical implica dividir una o más tablas en tablas más pequeñas, con el mismo número de registros, pero cada una de ellas con un subconjunto de columnas de la tabla original. Replicación y particionamiento se suelen usar combinados en el escalado de la capa de datos. A continuación vemos dos métodos de particionamiento horizontal en entornos multi-tenant.

5.4.1 Particionamiento horizontal basado en tenants

La forma más sencilla de escalar una base de datos multi-tenant es mediante el particionamiento horizontal basado en el ID del tenant. Las aplicaciones SaaS MT se adaptan bien a este tipo de particionamiento porque cada propietario tiene su identificador único y por tanto es fácil obtener el conjunto de registros de cada tenant.

Debemos tener en cuenta en el particionamiento horizontal que cada tenant tiene sus propias demandas de la aplicación y puede que uno sólo de ellos esté ocasionando la disminución del rendimiento. Por tanto un *scale-out* de este cliente no solucionaría su problema, si el del resto pero dejando particiones infrautilizadas.

Si experimentamos problemas en el rendimiento de la aplicación, debemos considerar no el número de tenants para una división equitativa, sino en el número de usuarios finales. Por ejemplo si tenemos en nuestra instancia de la aplicación siguiente situación de la Tabla 1, una solución sería mover los tenants C, D y E al nuevo servidor

equilibrando por tanto la carga en 1200 usuarios para cada servidor. Esta partición es la correcta para el caso de que tengamos muchas conexiones concurrentes a la base de datos. Sin embargo, si experimentamos problemas con el espacio ocupado por los registros y las consultas a la base de datos tardan un tiempo muy elevado, debemos por tanto pensar en ecualizar la carga de datos de cada uno de los servidores.

Tenant_ID	*Usuarios finales*
A	600
B	600
C	400
D	400
E	400

Tabla 1. Número de usuarios finales por cada tenant

Como se puede observar, la decisión del método de particionamiento a utilizar es especialmente importante. En cualquier caso, independientemente del método que escojamos, es clave que dispongamos de los herramientas de medición precisas para obtener las métricas sobre las cuales obtener una decisión.

5.4.2 Scale-out de un único tenant

En ocasiones, tanto el número de usuarios finales como la cantidad de datos almacenados, justifican el escalado de un único tenant a un servidor dedicado. Si la base de datos continua creciendo, llegará un momento que un único servidor no sea suficiente para este tenant y sea necesario hacer un scale-out. Escalar una base de datos dedicada es diferente al escalado de una compartida; es necesario analizar el tipo de datos almacenados y ver cual es el mejor enfoque. Algunos aspectos a considerar son:

- Uso de replicación de sólo lectura para datos que no cambian frecuentemente.
- Localización de los datos: Mantenimiento próximo de datos relacionados.
- Identificación de la información que no debe ser particionada.
- Uso de replicación single-master siempre que sea posible.

Capítulo IV

SaaS y la empresa

Este capítulo presenta un estudio del grado de penetración de SaaS en el mundo empresarial. Para ello, inicialmente se introduce el concepto de sistemas de información estudiando los tipos de más conocidos, sus funcionalidades y su relación con cloud computing. Posteriormente, se profundiza en el mundo de la pequeña y mediana empresa para ver la importancia y peso de este tipo de sociedades en todo el mundo. Finalmente, se analiza la relación de SaaS con este conjunto organizaciones, ponderando diferentes factores para su migración a aplicaciones en la nube.

1 Introducción

Los Sistemas de Información Empresarial (EIS, *Enterprise Information Systems*) emergen como una herramienta prometedora para la integración y mejora de los diferentes procesos de negocio [124]. Durante las últimas décadas son más y más las empresas que han acogido bien los EIS para llevar las riendas de su negocio [125].

En el año 1998 Davenport [126] llegó a afirmar: "*Los sistemas empresariales son un sueño hecho realidad… Mientras el crecimiento de Internet ha atraído la mayor parte de atención de los medios en los últimos años, en realidad la aceptación de sistemas empresariales en el mundo de los negocios sea quizás el avance más importante en el uso de las nuevas tecnologías dentro de las empresas en los años 90*"

En este modelo SaaS, no es necesario que las sociedades acometan inversiones iniciales de infraestructura hardware, personal o de desarrollo de la aplicación EIS; además, los costes operacionales se reducen de forma significativa ya que son distribuidos entre los diferentes arrendatarios.

Esto hace de SaaS un modelo atractivo para las empresas y en concreto para las pequeñas y medianas empresas (*pymes*) ya que por primera vez pueden acceder a soluciones empresariales de alta calidad.

Sin embargo, la adopción de SaaS por parte de las pymes está lejos de ser una realidad, existen factores que no propician la aceptación generalizada y serán estudiados en este capítulo.

2 Un poco de historia

Según [127], en los años 70 la visión de un único sistema de información integral era todavía un espejismo para las empresas que hacían uso de ordenadores; no se sabe si era por la escasa capacidad de procesamiento y lenguajes de programación o porque simplemente estaban contentos de trabajar sobre módulos funcionales separados. En su lugar, las empresas creaban "*islas de automatización*"; cada vez que una empresa necesitaba una nueva aplicación, se programaba un nuevo sistema de información el cual apenas se integraba con el resto de sistemas existentes (en ocasiones la integración era manual). En consecuencia, obtener informes que implicaran combinaciones de

información de las diferentes islas era una tarea propensa a errores. Además de este, existían otros problemas como la duplicidad de información e inconsistencia de la misma, elevados costes de mantenimiento y actualización, etc. Las organizaciones empezaron entonces a arrepentirse de no haber perseguido desde el principio de la creación de un único sistema de información unificado.

La década de los 80 supuso uno de los mayores cambios en el entorno IT de las empresas [87]. El paso hacia el modelo de computación de usuario final y las arquitecturas cliente-servidor permitió a las empresas separarse gradualmente de la estructuras centralizadas; estábamos en la era "mainframe". Los directores de departamento tuvieron la oportunidad de diseñar sistemas de información que se adaptaran más a sus necesidades. Sin embargo los EIS de muchas de las organizaciones empezaron a estar muy fragmentados y carecían de una gestión centralizada, lo cual hacía muy difícil su integración.

El sueño de la creación de una aplicación integrada no murió. En los años 80-90 emprendedores de todo el mundo empezaron a desarrollar paquetes software integrales que compartieran una única base de datos. Estos paquetes recibieron el nombre de *Enterprise Resource Planning* (ERP). En 1988 aproximadamente el 40% de las empresas con ingresos anuales superiores al billón de dólares habían implantado sistemas ERP [127].

A finales de los 90, los EIS fueron pasaron a ser reconocidos como la nueva "panacea" en el campo de los sistemas de información, especialmente para los gerentes que afrontaban las consecuencias negativas de la rápida proliferación de los sistemas de información en sus empresas [87]. Los proveedores empezaron a perseguir agresivamente la venta a empresas más pequeñas debido a que suponían un mercado más amplio [127].

Hoy en día, a pesar de los teóricos beneficios que supone esta renovación tecnológica, el proceso de migración no ha sido elegante y **muchas empresas han fracasado en la implantación** [127]. Los EIS ofrecen grandes beneficios, pero también son grandes los riesgos que se corren. No sólo son caros y difíciles de instalar, sino que atan de manos a los directores de departamento ya que imponen su propia lógica de negocio [126].

3 Definición de EIS

Existen múltiples definiciones de EIS en la literatura. Una de las más referenciadas es del artículo "*Putting the enterprise into the enterprise system*" [126] en donde se define a los EIS como paquetes software comerciales que prometen una integración trasparente de toda la información que fluye en una empresa.

En [127] el autor extiende el alcance de la definición; considera a los EIS como paquetes de software comerciales que permiten la integración de los datos orientados a operaciones y procesos de negocio en toda la empresa (y tal vez con el tiempo, a lo largo de toda la *cadena de suministro* entre organizaciones).

Los EIS ofrecen un único sistema que es fundamental para la organización y que asegura que toda la información pueda ser compartida entre las diferentes áreas; proporcionan una plataforma que permite a las empresas integrar y coordinar sus procesos de negocios [124].

Según una encuesta realizada en [128], los sistemas empresariales deben de proporcionar en una plataforma común la mejora de procesos, visibilidad de los datos, reducción de los costes operativos, aumentar la capacidad de respuesta a los usuarios y mejorar la estrategia de toma de decisiones.

De acuerdo a [129], *"una aplicación empresarial tiene como primer y fundamental objetivo llevar un seguimiento a la información relacionada con las empresa; después de todo es por ello por lo que se les llama 'Sistemas de Información'"*.

Los EIS suponen un nuevo modelo de computación corporativa que permiten sustituir las aplicaciones tradicionales, incompatibles entre ellas por una **solución totalmente integrada**.

4 Cloud computing y EIS

Las ventajas de los EIS tradicionales podemos combinarlas con las de cloud computing y el modelo SaaS [126]. De hecho, los servicios de cloud computing pueden ser clasificados teniendo en cuenta si son EIS u otros servicios IT [109]. Hasta ahora la división que hemos estudiado considera los niveles de servicio desde el punto de vista del proveedor (IaaS, PaaS, SaaS). En este capítulo orientado al software empresarial

podemos plantear una clasificación de los servicios de cloud computing desde el punto de vista del cliente. Desde esta otra perspectiva (Figura 57) podemos clasificar los servicios cloud en **orientados a aplicación** (e.g. CRM, ERP y SCM en modalidad SaaS) y los **orientados a infraestructura** (e.g. IaaS y PaaS).

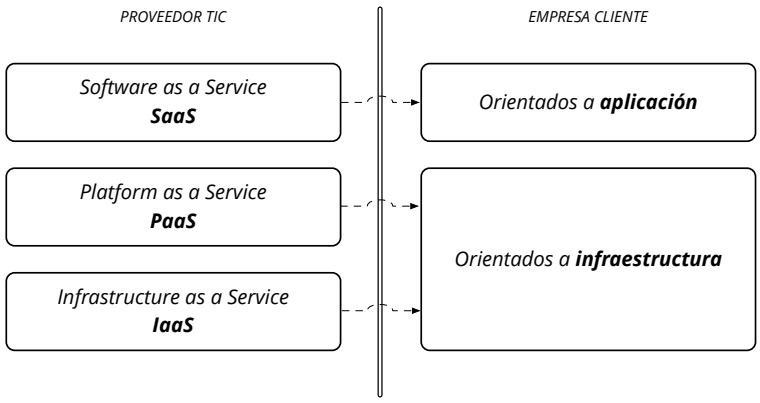


Figura 57. Clasificación de los servicios cloud desde el punto de vista del cliente

En el capítulo de SaaS se exponen de forma detalladas las ventajas (y desventajas) de SaaS para comercializar los EIS como SaaS, tanto desde el punto de vista del cliente, como del proveedor.

5 Tipos de EIS

Es la regla de las tres letras, todos conocemos las siglas pero pocos sabemos exactamente qué significan. CRM, ERP, CMS, DMS… ¿a qué corresponden? En las siguientes secciones se estudian diferentes tipos de aplicaciones empresariales, se descubre qué significan sus iniciales y a que uso están destinados. En este análisis, se hará más hincapié en aquellas más extendidas dentro de las empresas [129]–[131]: *Enterprise Resource planning* (ERP), *Supply Chain Management* (SCM) y *Customer Relationship Management* (CRM).

5.1 ERP

Las siglas ERP proceden del Inglés *Enterprise Resource Planning* que en su traducción literal significa: "Sistemas de planificación de recursos empresariales".

[132] considera que los ERP son una evolución de los MRPII (*Manufacturing Resource Planning*) y los define como "*soluciones software empresariales que buscan integrar el*

rango completo de los procesos de negocio, con el objetivo de proporcionar una visión global del negocio, desde una única arquitectura IT". Sin embargo, en este mismo artículo los autores hacen una análisis profundo para intentar llegar a una definición consensuada sin tener éxito: *"While ERP have gained some prominence in the IS literature over the past few years, we observe some dissent among academics on the nature and definition of ERP. Given the diversity of opinion illustrated above, it is unlikely that a broadly agreed upon definition of ERP can be achieved"*.

Según [133], los ERP buscan integrar los datos de inventario con la información financiera, ventas y recursos humanos permitiendo a las empresas dar precio a sus productos, generar informes financieros y gestionar los recursos de personal, materiales y económicos.

Existen varios autores que identifican los EIS con el término ERP II [124], [134] debido a la fuerte relación que tienen con los *Enterprise Resource Planning* (ERP); incluso, consideran que la siguiente generación de EIS será conocida como *Entire Resource Planning* (ERP) o *Complete Resource Planning* (CRP) [124]. El objetivo de este trabajo es la presentación de los sistemas empresariales para comprender mejor el grado de penetración de sus versiones SaaS en las empresas. Por tanto, no profundizaremos en más detalle.

En nuestro caso entendemos que los ERP integran la información de otros sistemas en la empresa (CRM, SCM, etc.) para el tratamiento de la información en una visión conjunta y con determinados fines. Podemos decir que los ERP intentan unifican los EIS individuales en uno corporativo de visión global.

El más conocido proveedor ERP es SAP [135]. SAP está considerada como el tercer proveedor independiente de software del mundo (tras Microsoft y Oracle) y el mayor de la Unión Europea. SAP cuenta con más de 260.000 clientes en 180 países y el año 2013 contó con unos ingresos totales de 16,82 billones de euros [136].

5.2 CRM

La palabra CRM proviene del Inglés *Customer Relationship Management* y se entiende como un programa para la gestión y mejora de la Relación con los Clientes.

Para [137] un CRM es una combinación de gente, procesos y tecnología que tiene como finalidad comprender los clientes de la empresa. Un CRM es un enfoque integrado de gestión, centrándose en el cliente y en la relación que mantenemos con él. Las compañías que implementen con éxito un CRM se verán beneficiadas de la fidelización del cliente y beneficio durante mucho tiempo.

Las soluciones de CRM permiten darle seguimiento a las actividades de los clientes. Nos ayudan a desarrollar un perfil para cada cliente de tal forma que podamos plantearle una oferta realmente hecha para él.

Las herramientas CRM maximizan las conversiones de cliente potencial a cliente real y buscan obtener el mayor *retorno de inversión* de las empresas (ROI). Se centran en áreas relacionados con la atención al cliente, gestión comercial, marketing y ventas (Figura 58).

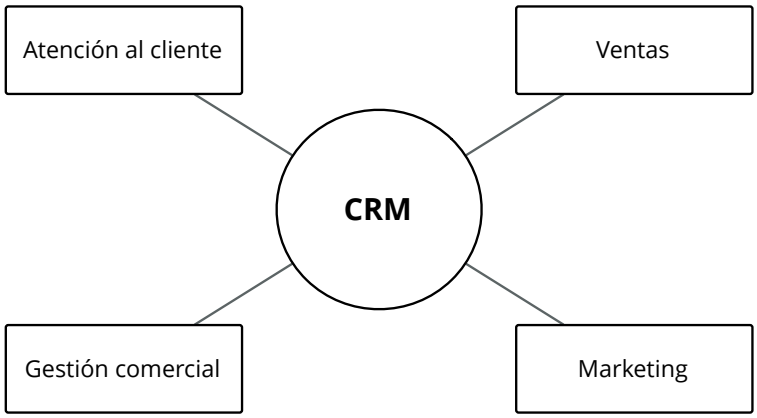


Figura 58. Áreas funcionales del CRM

En la actualidad, el líder del mercado proveedor de CRM comercializado como SaaS es Salesforce.com [53]. Existen CRM verticales también especializados de sector, como un CRM Inmobiliario, NetPropertyAgent.com [138].

5.3 SCM

La Gestión de la Cadena de Suministro (SCM*, Supply Chain Management*) es la tarea de integrar las diferentes unidades organizativas a lo largo de un suministro coordinando materiales, información y los flujos financieros. Tiene por objetivo

completar de forma satisfactoria las demandas de los clientes y mejorar la competitividad de la cadena como un solo elemento [139].

Los sistemas SCM se encargan de las órdenes de compra, emitir los pedidos, controlar el inventario de materias primas y de productos terminados, así como de enviarlo a los clientes [131]. Son herramientas para la gestión de la demanda de materias primas y el suministro de productos terminados y utilizadas por todas las empresas que intervienen el proceso (transporte, logística, proveedores de materia prima, etc.).

Este tipo de sistemas llegan a mejorar enormemente la producción y aumentar el beneficio de las empresas. Por ejemplo tenemos el caso de Dell [140]; antes de la implantación de un sistema SCM la compañía estadounidense tenía almacenes en los que se tardaban 20-25 días en realizar inventario. Hoy en día Dell no tiene almacenes y aun así monta cerca de 80.000 ordenadores cada 24 horas. La enorme cadena de suministro de la empresa está constantemente en funcionamiento.

5.4 ERP,CRM,SCM: Una visión conjunta

Existen solapamientos en una segregación funcional de los diferentes sistemas [131] (ver Figura 59). Por ejemplo, la que se da entre los pedidos (SCM) y facturación (ERP) que a su vez tiene una fuerte relación con los CRM.

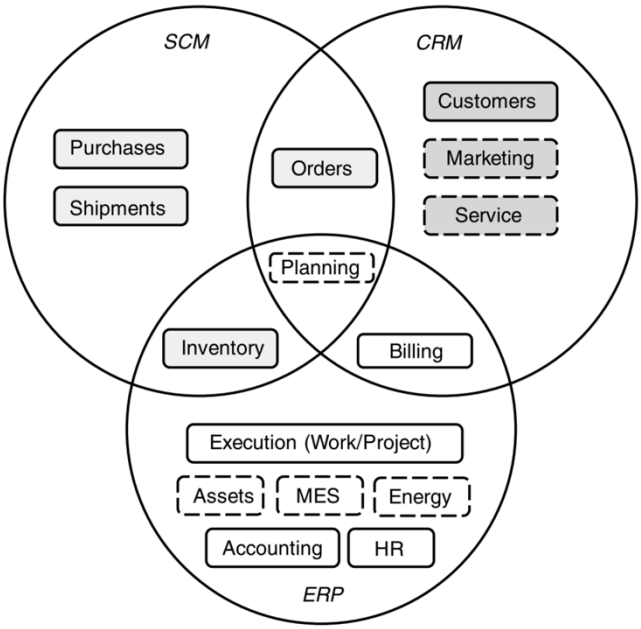


Figura 59. Segregación funcional de un EIS

Cuando se contrata un sistema, es importante saber si dispone de este tipo de interrelaciones ya que si no puede llevar a la redundancia o requerir nuevas funcionalidades que conlleven un aumento de precio.

5.5 Otros tipos de EIS

- **CMS**, Un Sistema de gestión de contenidos (*Content Management System* en inglés, abreviado CMS) es un programa que permite crear una estructura de soporte para la creación y administración de contenidos, principalmente en páginas web y por parte de los usuarios. Los CMS permiten que cualquier usuario sin conocimientos de programación ni maquetación pueda gestionar contenido en el portal web. Aunque no podemos categorizarlo como software puramente empresarial, debido a que está destinado a uso general, si es interesante conocer el significado para el objetivo de esta sección. Entre los CMS más conocidos y utilizados nos encontramos con *Drupal* [141] o *Joomla* [142].

- **DMS,** *Document Management System* consiste en la gestión documental de archivos. Sistema informático utilizado para rastrear y almacenar documentos electrónicos e imágenes de documentos en papel. Suele proporcionar el almacenamiento, la seguridad y las capacidades de recuperación e indexación del contenido.

- **ECM**, *Enterprise Content Management*. Fue definido en el año 2000 por la AIMM (*Association for Information and Image Management*) [143]. Un EMS es un sistema que modela las estrategias, métodos y herramientas utilizadas para capturar, gestionar, almacenar, preservar y proporcionar contenidos y documentos relacionados con procesos corporativos. Un ECM cubre el conjunto completo de información de un empresa, tanto si es un documento electrónico, un impreso parcial de resultados de base de datos o incluso un email.

- **EAM**, *Enterprise Asset Management*. Gestión de los activos físicos de una empresa (maquinaría, etc.). Incluye aspectos como el diseño, construcción, mantenimiento y reparación. Es un sistema de gestión evolución de los MRO (*Maintenance, Repair & Operations*) y CMMS (*Computerized Maintenance Management Systems*).

- **MRP**, *Material Requirements Planning* (planificador de necesidades de materiales). Sistema de Información encargado de la gestión del material necesario para la producción, que asegura inventario en todo momento.

- **PLM**, *Product Lifecycle Management* (Gestión del Ciclo de Vida del Producto) Estos sistemas se ocupan del ciclo de vida completo del producto, desde si diseño y fabricación, hasta su servicio y eliminación.

- **PRM,** *Partner Relationship Management* (Gestión de las relaciones con colaboradores). Análogo a los CRM, se trata de un sistema de Información que mejora la colaboración con colaboradores y otras entidades.

- **SRM,** *Supply Relationship Management* (Gestión de las relaciones con proveedores). PRM específico para proveedores.

6 La pequeña y mediana empresa

Las pequeña y mediana empresa (pyme) es columna vertebral en cualquier fase de desarrollo económico. Ahora mismo estando en época de crisis, vemos como son los pequeños empresarios los que están consiguiendo eliminar la recesión. En la mayoría de los países, las pymes suponen más de un 50% del PIB (*Producto Interior Bruto*) [144]–[146]. Por otro lado, el poder disponer de un software asequible y de calidad es de suma importancia para la eficacia eficiencia de la empresa moderna. Como vimos antes, cloud computing y en concreto su modelo SaaS de pago por uso de software plantean una posible solución a esta necesidad.

Las aplicaciones son accedidas vía web y son propiedad de las empresas de software. Los usuarios pasan de propietarios a arrendatarios (en inglés, *tenants*) de los sistemas, bajo un contrato de condiciones de uso (SLA) en donde se fijan, entre otros, la frecuencia y la cuota asociada de pago (normalmente mensual o anual). Tal y como podemos observar en la Figura 62, se eliminan las inversiones iniciales (CAPEX, *Capital Expenditures*) al estar las soluciones instaladas en los centros de datos y no ser necesarias inversiones en infraestructura, compra o desarrollo del sistema. Gracias a la arquitectura multi-tenant, los costes operacionales (OPEX, *Operational Expenditures*) de uso y mantenimiento de la aplicación también se recortan drásticamente ya que son distribuidos entre las diferentes empresas suscriptoras.

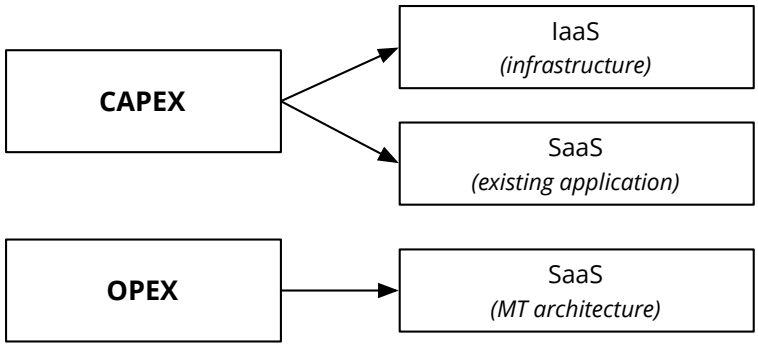


Figura 60. Reducción de costes de adquisición del software

Esta simplificación y reducción de costes hace posible el acceso de las pymes a soluciones EIS hasta ahora inasequibles y reservadas sólo para las grandes corporaciones [9].

Sin embargo, el rango en el cual se encuadran las pequeñas y medianas empresas es muy variable; empresas con 200 empleados son consideradas como pymes pero también negocios con 10; por no hablar de los trabajadores autónomos, que también entran dentro de este conjunto. Este último tipo de *micro-empresas* de pocos trabajadores puede que no dispongan de los recursos económicos suficientes como para beneficiarse de SaaS. Es más, incluso teniéndolos quizás no les interese su uso ya que estas aplicaciones desarrollan una lógica demasiado compleja para sus negocios.

Tal y como veremos posteriormente, las micro-empresas suponen el 80% del total de las empresas. Si las pymes no pueden beneficiarse de esta nueva fórmula (bien por motivos económicos, bien por no ser adecuadas las aplicaciones), el clamor de una "computación para todos" establecido por la visión de la 5ª utilidad no prevalecerá.

6.1 El papel de las pymes en el mundo

Según la Comisión Europea (CE) [147], las pequeñas y medianas empresas (pymes) juegan un papel fundamental en la economía Europea, Una empresa es "*Cualquier entidad ligada a una actividad económica, independientemente de su naturaleza legal*", por tanto, los autónomos encajan dentro de esta definición. La siguiente tabla muestra la categorización de empresas:

Categoría	*Empleados*	*Vol. de ventas anual*	*Balance anual*
Mediana	< 250	≤ €50 millones	≤ €43 millones
Pequeña	< 50	≤ €10 millones	≤ €10 millones
Micro	< 10	≤ €2 millones	≤ €2 millones

Tabla 2. Clasificación de empresas según la CE

Más recientemente, en su *informe anual sobre la pequeña y mediana empresa (2011-2013)* [148], la CE estimó la existencia de 20.7 millones de micro empresas en la UE. Dentro de esta grupo, una aplastante mayoría del 92.2% son micro-empresas; el 6.5% son "pequeñas" y sólo el 1.1% pertenecen al grupo de los "medianos".

Por tanto, el conjunto de pyme representa el 99.8% del total de las empresas en Europa mientras que sólo el 0.2% son grandes empresas. En términos de empleo, el grupo de pymes proporciona 87 millones de puestos de trabajo suponiendo el 67,4% del total.

En EEUU, la Comisión Internacional de Comercial (*United States International Trade Commission, USITC*) define empresa como [149]*: "an individual proprietorship, partnership, joint venture, association, corporation (including any subsidiary corporation), business trust, cooperative, trustee in bankruptcy, or receiver under decree of any court"*.

En la clasificación americana, el conjunto de pequeñas y medianas empresas (SMES, *Small and Medium Enterprises*) incluye a aquellas sociedades con menos de 500 empleados. La información de la Tabla 3 fue extraída de la oficina del censo de EEUU (*U.S. Census Bureau*) [150]; en ella se estima que el 99,7% de las empresas son pymes y que dentro de esta categoría el 79,2% tienen menos de 10 empleados. Por tanto, casi 4/5 de las empresas en EEUU son micro-empresas. EL porcentaje crece hasta un 90% cuando incluimos en este cálculo empresas de menos de 20 trabajadores.

Categoría	*Empleados*	*Número*	*Porcentaje*
Pequeñas (pymes)	0-4	3.575.240	62,3%
	5-9	968.075	16,9%
	10-19	617.089	10,8%
	20-99	475.125	8,3%
	100-499	81.773	1,4%
	Total pymes	5.717.302	100%
Grandes empresas	500+	17.236	0,3%
	Total	11.451.840	200%

Tabla 3. Clasificación de empresas según la USITC

Este escenario no sólo ocurre a nivel europeo o norte-americano, sino que también en el resto del mundo. En Latino-América, el 80-90% de las empresas son micro-empresas; en Asia, el 81% de los puestos de trabajo es proporcionado por las pymes mientras en el Sudáfrica se estima en un 60% [151].

Por tanto, a nivel mundial existe una mayoría aplastante de pymes y su importancia en el crecimiento de la economía global es un hecho. Dentro de este porcentaje la media de micro-empresas es muy elevada, con porcentajes en torno al 80%-90%.

6.2 La pyme en Andalucía

Según [152], al adentrarnos en el estudio del tejido empresarial de nuestra comunidad autónoma nos encontramos con una característica absolutamente definitoria, su *atomización*. Las estadísticas hablan por si solas, el 99.9% de nuestras empresas tienen menos de 250 trabajadores en sus plantillas, y podemos afinar aún más : el 90% posee menos de 10.

Las pymes en Andalucía absorben a más del 80% del empleo total, luego las actuaciones de Política Económica que vayan encaminadas a promocionar a este colectivo tendrán una repercusión inmediata en nuestras preocupantes cifras de desempleo.

En [153] un informe analiza más en detalle la segmentación de empresas en Andalucía. De las 481.926 empresas de la Comunidad Autónoma, 481.620 son pyme (0 a 249 asalariados), lo que supone el 99,94% del total de las empresas de esta comunidad autónoma. El 96,14% son microempresas (0 a 9 asalariados) de las cuales el 53,88% son empresas sin asalariados. El número de grandes está muy por debajo de la media nacional: 0,06% frente 0,12%. En este grupo son las provincias de Almería y Sevilla, las que cuentan con una mayor participación ya que el 0,09% de sus empresas tienen 250 o más asalariados, le siguen Huelva y Granada con el 0,07% y el 0,05%.

7 Pymes y SaaS

El mercado de soluciones SaaS empresariales es muy dilatado; ya hemos analizado típicos ejemplos de EIS como los CRM, ERP o SCM. Si una empresa pyme quiere migrar sus aplicaciones a la nube, encontrará un amplio abanico de posibilidades.

El intervalo de las pequeñas empresas es muy grande (de 1 a 200 empleados en Europa y hasta 500 en EEUU) y en él podemos encontrar una gran variedad. Según [151] *"el problema de las pymes es cuestión de escala. Una empresa con 20 personas es considerada igual que otra con 500; la cual a su vez es también vista como una de 5000"*.

En consecuencia y según los datos ofrecidos anteriormente, no podemos esperar que un negocio de cinco personas tenga las mismas necesidades funcionales que uno de doscientos, del mismo modo que no tienen equiparables recursos ni complejidad.

Las empresas pequeñas situadas en el extremo del intervalo están todavía lejos de implantar este modelo; es más, muchas de ellas todavía no tienen implantado modelo alguno. Por tanto, el principal problema radica en que agrupamos dentro del mismo conjunto empresas que tienen distintas necesidades funcionales de software.

7.1 Factores para la migración a SaaS

La **reducción de costes** es considerada como una de las principales razones de migración al cloud por parte de las pymes. En un estudio reciente [24], se ha puesto de manifiesto que la fórmula actual de precios hace de cloud computing una tecnología altamente adecuada para las empresas. En su encuesta "*An SME perspective on cloud computing*" [25], la *European Network and Information Security Agency* (ENISA) se descubre que la mayor parte de las empresas preguntadas (68,1%) piensan que una de las principales razones para el cambio al cloud es la eliminación de los gastos de inversión inicial (CAPEX). Existe también otra encuesta realizada sobre empresas del Reino Unido [26] en la que el ahorro económico es considerado como uno de los principales promotores de la migración al cloud.

A pesar de esta bajada de precios, SaaS no está extendido en las empresas como se podría esperar. Hablando en un sentido amplio podemos decir que las empresas mas grandes son más propensas a beneficiarse de sistemas de información en la nube [27], mientras que las empresas más pequeñas todavía lo pueden encontrar estas aplicaciones inasequibles o inadecuadas.

Supongamos una empresa con cinco empleados que está dispuesta a suscribirse a un CRM en la nube. La Tabla 4 muestra una comparativa de previos de los principales proveedores en Mayo de 2014 (sacada de las páginas web de las diferentes empresas). Los precios mensuales están multiplicados por el número de usuarios; algunos de ellos por el total de la inversión, ya que obligan a una inversión inicial de al menos un año. Se estima también el coste de un uso por espacio de dos años.

PROVEEDOR	PRODUCTO	USUARIO/MES	TOTAL MES	FACTURACIÓN	INVERSIÓN	COSTE A DOS AÑOS
SALESFORCE	SALES CLOUD	$65,00	$325,00	ANUAL	$3.900,00	$7.800,00
SAP	SAP SALES ON DEMAND	$80,00	$400,00	MENSUAL	$400,00	$9.600,00
ORACLE	ORACLE CRM ON DEMAND	$70,00	$350,00	ANUAL	$4.200,00	$8.400,00
MICROSOFT	MICROSOFT DYNAMICS	$44,00	$220,00	MENSUAL	$220,00	$5.280,00
SUGARCRM	SUGARCRM	$35,00	$175,00	ANUAL	$2.100,00	$4.200,00
SAGE	SAGE CRM CLOUD	$39,00	$195,00	ANUAL	$2.340,00	$4.680,00

Tabla 4. Cuadro comparativo de precios de los principales proveedores de CRM en la nube

En un análisis más profundo, si estudiamos la Figura 61 la inversión inicial varia claramente dependiendo del criterio de facturación del proveedor. Las soluciones de Microsoft o SAP podrían implantarse pagando cantidades "asequibles" inferiores a 500$ (permiten cuotas de suscripción mensuales). Sin embargo, debemos de considerar un periodo más amplio, ya que la valía y eficacia de un EIS no puede ser comprobados en un par de meses. Si analizamos el coste de SAP en un periodo de dos años, comprobamos que uno de los sistemas más económicos inicialmente, terminan convirtiéndose en el servicio más caro de todos. *SugarCRM* destaca como el más barato de todos (cercano a *Sage*), pero implica un pago inicial de más de 2000$ por la suscripción anual.

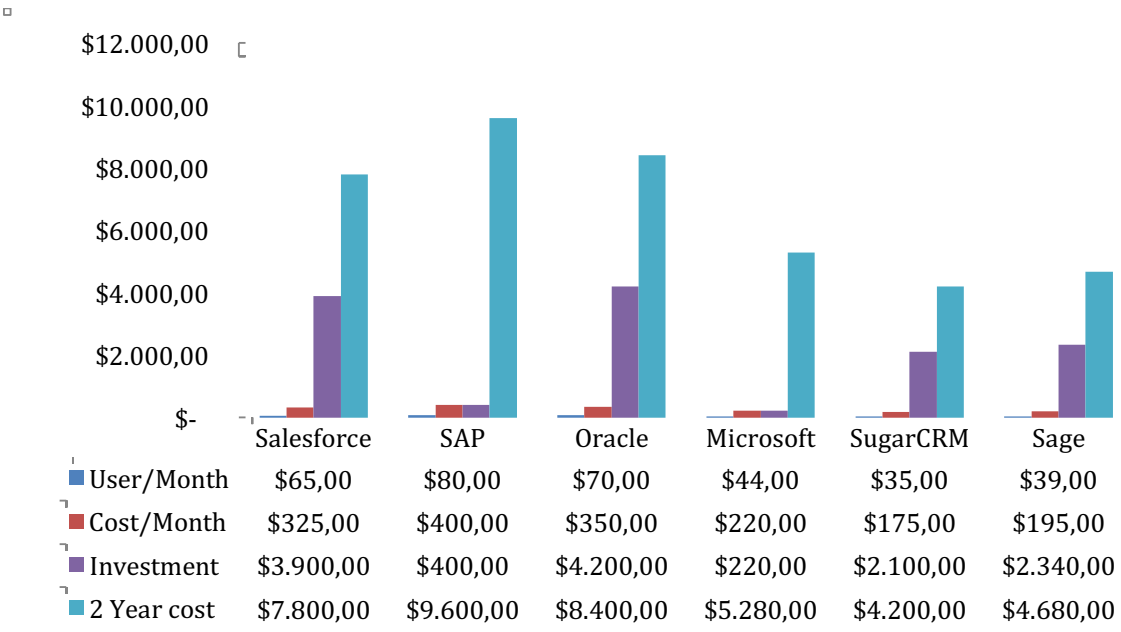


Figura 61. Comparación de precios mensuales. inversión inicial y coste a dos años

Todo ello teniendo en cuenta que los precios son por usuario; los costes se disparan para adquisiciones de más de una licencia. Además, existen **costes ocultos** que

desconocemos pero pueden surgir con el uso (cuota de disco, ancho de banda, etc.). Estos gastos inesperados pueden ser sustanciosos e inevitables debido a la dependencia que se tiene una vez implantada la aplicación [7].

La **personalización de la aplicación** puede ser también un escollo [154], [155]. Los grandes proveedores SaaS permiten vías de personalización de forma que los tenants puedan tener experiencias únicas de usuario. En las aplicaciones SaaS genéricas horizontales [87], los clientes pueden ver satisfechas las necesidades de su industria y de su empresa al personalizar el modelo de datos, adaptar la capa de presentación o modificar el flujo de trabajo de la aplicación [14] gracias a los metadatos.

En el caso de una pequeña empresa, los departamentos de informática son escasos y estas modificaciones implican un conocimiento técnico que lo más probable es que la empresa no tenga. Además, encargar este trabajo a terceros o al proveedor SaaS implica una nueva subida del coste.

Por ejemplo, un consultor SAP cobra entre los 96€ por hora de un consultor asociado hasta los 330€ del jefe de consultoría [156]. Dadas estas circunstancias una empresa puede que contrate SAP pagando los 400€ iniciales que supone la primera cuota (Figura 61), pero los costes ocultos que supone el trabajo de consultoría para adaptar la aplicación a las necesidades de la empresa puede que terminen minando la continuad de la suscripción, con la consiguiente pérdida de tiempo y dinero.

Según [157], a pesar de todas las ventajas que ofrece esta tecnología, **no todas las empresas se están apresurando a su implantación**. Las soluciones cloud todavía no han alcanzado la madurez suficiente debido al alto nivel de riesgos relacionados, los costes ocultos, la ausencia de aplicaciones específicas de industria y el modesto gasto en tecnología que acometen las empresas. Si se le pide a una pequeña empresa que se gaste tal cantidad de dinero, probablemente termine volviendo a sus soluciones on-premises tradicionales (si es que tuviera alguna).

Dejando aparte los factores económicos y de conocimiento tecnológico, la **dificultad en el aprendizaje** es otro factor que se debe tener en cuenta [158]. En teoría, las aplicaciones empresariales tienen como objetivo reducir esfuerzos y aumentar el rendimiento. Los programas financieros por ejemplo, agilizan notablemente los

procesos de contabilidad y aún así, no siempre están presentes en pymes. De acuerdo a [159], las empresas pequeñas además de tener un presupuesto muy reducido en software en comparación con las grandes compañías, tienen unas necesidades funcionales mucho menores a las suministradas por los grandes proveedores SaaS. Los requisitos de la empresa podrían ser cubiertos por tan sólo unas pocas características de los programas comerciales.

Por otro lado debemos contemplar la *utilidad* de la aplicación. Incluso si una pequeña pyme es muy rentable y se puede permitir el pago, estos programas están **normalmente preparados para empresas más grandes** y presentan demasiadas opciones y funcionalidades que resultan complejas e innecesarias. Requisitos simples puede que no estén presentes y por lo tanto es necesario añadirlos con el consiguiente aumento del precio. Por otro lado, existe otra gran cantidad de funcionalidades que el sistema ofrece y que no serán utilizadas pero que sin embargo son pagadas.

La tecnología está destinada a simplificar los negocios [160] y el principal objetivo del software empresarial debería ser reducir el esfuerzo y hacer más fáciles las tareas diarias de la empresa, dando soporte a los diferentes procesos de nuestro negocio y no al contrario. Como resultado, muchas empresas toman el camino de *hazlo-tu-mismo* (DIY, *Do It Yourself*) mediante el uso de hojas de cálculo o anotaciones. En muchas ocasiones, no es una cuestión de trasladar las aplicaciones tradicionales al cloud, es cuestión de tener algún tipo de aplicación; un software que además guíe a la empresa en cómo hacer las cosas (ya que proporciona la lógica de los procesos de negocio), centralice la información y elimine el caos de hojas de cálculo de la oficina.

La **localidad y cercanía del proveedor** de servicios está altamente relacionado con la privacidad y la preocupación en la seguridad de los datos [26], [108], [109]. Ya que no pueden tenerlos ellos, los clientes prefieren confiarle los datos a un proveedor más familiar. Asimismo, estos proveedores conocen a la perfección a su cliente y pueden ayudar y dar soporte de una forma rápida; por el contrario, las grandes empresas, con call centers anónimos de llamadas tediosas e interminables pueden terminar con la paciencia de los suscriptores [161].

Sin embargo, encontrar un proveedor SaaS que sea además una pequeña pyme local es complicado o más bien imposible (podríamos atrevernos a decir que no existen). El

éxito de este modelo radica en las economías en escala y en la comercialización al Long Tail del mercado del software [22], [23]; para ello, es necesaria una fuerte inversión en marketing y publicidad, lo cual supone unos recursos económicos que los estos proveedores no tienen.

Debido a la escalabilidad, los sistemas multi-tenant necesitan unos **recursos humanos** superiores a las diez personas que marca el umbral de las micro-empresas. En la actualidad, la actividad de las empresas locales de software se reduce a la venta de licencias de paquetes software on-premises tradicionales. En otros casos estas empresas contratan *freelances* (también normalmente locales) para la programación de aplicaciones a medida de sus clientes; soluciones que suelen ser **específicas de industria** y por supuesto sin capacidad multi-tenant. Incluso en el raro caso de que un proveedor local ofrezca un servicio con capacidad de escalamiento multi-tenant, el número de clientes potenciales no será suficientemente elevado como para poder mantener el sistema rentable. Las empresas chicas tienen el espectro de oportunidades comerciales limitado no sólo por la funcionalidad ofrecida (a una sola industria), sino que también por el rango operacional que está reducido geográficamente a las empresas cercanas.

En resumen, SaaS es una fórmula muy atractiva para las pymes ya que supone una reducción de costes iniciales y operacionales elevada. Sin embargo, **la adopción de SaaS en este conjunto de empresas no ha llegado a cuajar.** El intervalo de las pymes es muy amplio y las necesidades software de todas las empresas que se consideran en el mismo no debe de ser unificado. El 80% de las pymes son micro empresas con menos de diez trabajadores que bien no pueden pagar SaaS al ser todavía caro, bien no les sirven las soluciones por ser demasiado complicadas. Proveedores pequeños y locales por su parte tiene un radio de alcance muy limitado y no pueden arriesgar en el negocio del SaaS ya que supone inversiones en marketing para la difusión de un servicio cuyo éxito depende en la escalabilidad.

7.2 Vendedores SaaS

Pasar de un modelo de comercialización tradicional (*on-premises*) al modelo SaaS requiere un cambio importante en tres áreas interrelacionadas: el modelo de negocio, la arquitectura de la aplicación y estructura operacional (Figura 62). En [23] podemos ver

un amplio desarrollo de dichos cambios; en este trabajo introduciremos los aspectos más importantes.

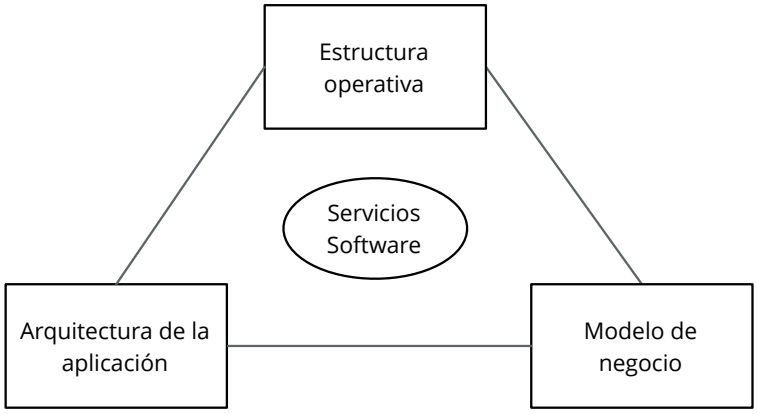


Figura 62. Áreas en las que el vendedor de software debe modificar su forma de pensar

Cambiar el *modelo de negocio* supone realizar una o varias de las siguientes modificaciones:

- **Trasladar la propiedad del software** del comprador a un proveedor externo.
- La **responsabilidad del mantenimiento** de la infraestructura tecnológica y la gestión (hardware y técnicos profesionales) pasa de estar del lado del consumidor a depender del proveedor de software.
- **Reducción del coste de producción** de software, aplicado economías de escala.
- **Cambiar el target cliente** al *long tail* de las pequeñas empresas (pymes), reduciendo al mínimo el coste del software.

Como podemos observar, el cambio implica la modificación de formas de pensar tanto del cliente como del proveedor. Es responsabilidad del proveedor el persuadir al cliente para realizar este cambio.

En segundo lugar, los proveedores deben modificar la *arquitectura de su aplicación* si desean comercializar sus soluciones como SaaS. Con este nuevo modelo, las soluciones deben de ser compartidas entre los diferentes suscriptores y esto se consigue gracias a la eficiencia multi-tenant de las aplicaciones. En este trabajo hemos dedicado un capítulo completo a las AMTs y no nos detendremos más en este aspecto.

El tercer cambio importante está relacionado con *la estructura operacional*: qué es necesario hacer para llegar a los clientes la aplicación, para mantenerla disponible y

funcional a un coste adecuado. Para muchos proveedores, quizá el cambio más notable sea la necesidad de gestionar data centers, ya que hasta ahora no les había hecho falta. El software tradicional no requiere de estos conocimientos ya que se instala localmente, sin embargo, con SaaS deben de convertirse en verdaderos expertos en la materia.

Capítulo V

Agilidad y Arquitectura

En este capítulo se resume el marco teórico existente para *agilidad* y la tendencia actual para aplicarla a nivel arquitectónico. Primero, se profundiza en el *manifiesto ágil* como fuente de nacimiento de la agilidad, en sus valores y principios. Segundo, repasamos las metodologías ágiles existentes más importantes así como la filosofía *lean* para el desarrollo de software. Por último, nos adentramos en la reciente discusión para alinear dos conceptos hasta ahora enfrentados, agilidad y arquitectura.

1 Introducción

En la década de los 90 la filosofía de desarrollo que primaba por aquel entonces se podía resumir en *"Hazlo bien desde la primera vez"*) [162]. Era un pensamiento generalizado para los desarrolladores que los proyectos no irían bien si no se ceñían a una metodología estricta desde el primer momento. Si no existía una planificación pesada y escrupulosa inicial, los proyectos estaban abocados al fracaso. La realidad es que con este tipo de planificación rara vez salían bien los proyectos. Al mismo tiempo, existía un trasfondo latente de una corriente alternativa que había empezado a socavar esta doctrina consiguiendo resucitar proyectos y siguiendo una metodología más liviana, basada en iteraciones. Había nacido el *desarrollo ágil de software* (*Agile Software Development*, ASD.

La *agilidad* ha sido ampliamente defendida en los últimos años como una filosofía que mejora la eficiencia en la construcción del software [17]. La mayoría de las metodologías ágiles se centran en la organización de los miembros del equipo y en el uso de prácticas demostradas de la Ingeniería del Software como el *refactoring* [14], sin prestar atención a la arquitectura. Los defensores de estas técnicas de desarrollo consideran que la arquitectura supone un pérdida de tiempo, que acarrea grandes cantidades de documentación y diseño inicial sin utilidad.

Sin embargo, en un ambiente cloud multi-tenant donde la facilidad para la escalabilidad es clave, la existencia de arquitecturas que den soporte a la agilidad y un rápido aprovisionamiento es crítico. Según [163], "*la adopción de SaaS crece y evoluciona en los mercados de software empresarial*", y es que, cada vez son más las empresas que están adoptando esta nueva fórmula. En aplicaciones SaaS MT la demanda debe de ser cubierta por estilos arquitectónicos que aceleren la configuración de las suscripciones de los clientes" [14]. Las AMTs deben facilitar "*frameworks administrativos que mejoren la eficiencia en la gestión para administrar el sistema*" [15].

Las arquitecturas cloud para SaaS no deben de ser consideradas como "malignas" en la ingeniería del software, sino como herramientas de ayuda y activos de gran soporte para los equipos de desarrollo ágil.

2 Orígenes

El *Desarrollo Ágil de Software* (ASD) no es un conjunto de herramientas ni una única metodología, sino una filosofía a seguir que fue plasmada en el manifiesto *Agile Manifesto* [164]. El documento fue firmado en el año 2001 por un conjunto inicial de 17 personas convocados por *Kent Beck*, quien había publicado un par de años antes *Extreme Programming Explained* [165]. La reunión tuvo lugar en Salt Lake City (Utah) para tratar sobre técnicas y procesos en el desarrollo de software.

Críticos con las metodologías tradicionales (metodologías más bien *pesadas*, basadas en la formalidad y en la documentación exhaustiva) resumieron los valores sobre los que se basan los métodos alternativos de desarrollo en *cuatro postulados* (los veremos más adelante). Además, establecieron *doce principios* en los que basarse para un desarrollo ágil. Al conjunto de personas que lo firmaron se les conoce con el nombre de *Alianza Ágil* (*Agile Alliance* [166]).

La aplicación de esta filosofía ha dado lugar a la aparición de varias *metodologías ágiles*: *Extreme Programming (XP), Scrum, Adaptive Software Development (ASD) o Crystal Methodologies* entre otras. Estas metodologías dan mayor valor al individuo, a la integración del cliente, y a entregas cortas de software funcional. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad [167].

Esta filosofía de desarrollo tiene su bases originales en otras metodologías para procesos industriales (distintos al desarrollo de software) de finales de los años 1940; el precursor fue en la empresa japonesa de automoción Toyota y es conocida con el nombre de *Lean Development* (LD). En entornos de desarrollo de software, este modelo se conoce como *Lean Software Development* [168].

Agile supone una separación importante de los métodos tradicionales basados en documentación, como por ejemplo *Waterfall* (cascada).

Hoy en día existe una gran aceptación de este nuevo movimiento. Según un estudio realizado en el año 2006 [169], el 41% de los proyectos ya habían adoptado metodologías ágiles y el 65% de dichos proyectos usaban técnicas ágiles. En el año

2011, un 40% de las empresas americanas ya estaban siendo ágiles [170]. De acuerdo con [171], los beneficios que suponen las metodologías ágiles han sido reconocidos por la industria IT.

3 El Manifiesto Ágil

En Febrero del año 2001, diecisiete analistas software de prestigio y amplia experiencia se reunieron para hablar sobre el proceso de desarrollo de software, aprobando la redacción de un manifiesto acordado de forma unánime. Todos los asistentes lo rubricaron a pesar del número y de pertenecer a empresas diferentes e incluso competidoras. El *Manifiesto Ágil* [164] firmado por 17 desarrolladores, comienza así:

Estamos descubriendo formas mejores de desarrollar
software tanto por nuestra propia experiencia como
ayudando a terceros. A través de este trabajo hemos
aprendido a valorar:

- *A los **individuos y su interacción**, por encima de los procesos y las herramientas.*
- *El **software que funciona**, por encima de la documentación exhaustiva.*
- *La **colaboración con el cliente**, por encima de la negociación contractual.*
- *La **respuesta al cambio**, por encima del seguimiento de un plan.*

Esto es, aunque valoramos los elementos de la derecha,
valoramos más los de la izquierda.

Según el manifiesto, los procesos, herramientas, contratos y documentación son útiles. Sin embargo, hay muchos momentos en los que priman la presión y los resultados y es entonces cuando debemos ponderar la importancia de todos los aspectos y poner unos por encima de otros.

El movimiento ágil no condena el uso de un método, de hecho muchos de los integrantes de la Alianza quieren restaurar la credibilidad del concepto buscando un equilibrio. Son partidarios de aspectos relacionados con el desarrollo tradicional pero con ciertos matices:

1. *Defienden el modelado*, pero no para almacenar los diagramas en un almacén de la empresa sin darle utilidad alguna.

2. *Defienden la documentación*, pero no cientos de páginas nunca actualizadas y tomos rara vez usados.
3. *Defienden la planificación*, pero entienden los límites de la planificación en un entorno turbulento y cambiante.

3.1 Valores

El artículo [172] recuerda y analiza los valores del especificados en el manifiesto:

Los individuos e interacciones por encima de los procesos y las herramientas

Para garantizar una mayor productividad, las metodologías ágiles valoran el equipo humano como el principal factor de éxito. Reconocen que contar con equipo cualificado con capacidades técnicas adecuadas, facilidades para adaptarse al entorno y trabajar en equipo así como de interactuar convenientemente con el usuario, da mayor garantía de éxito que contar con herramientas y procesos rigurosos. Las metodologías ágiles reconocen que es más importante construir un buen equipo de trabajo que las herramientas y procesos.

Software funcional por encima de la documentación

Los profesionales relacionados con el desarrollo de software, aunque no es su fuerte producir documentos, reconocen su importancia; al igual que reconocen el tiempo y costo de mantener una documentación completa y actualizada.

En este sentido, las metodologías ágiles respetan la importancia de la documentación como parte del proceso y del resultado de un proyecto de desarrollo de software. Sin embargo, con la misma claridad hacen énfasis en que se deben producir los documentos estrictamente necesarios; los documentos deben ser cortos y limitarse a lo fundamental, dando prioridad al contenido sobre la forma de presentación.

La documentación en las metodologías ágiles hace uso de mecanismos más dinámicos y menos costosos, como son la comunicación personal, el trabajo en equipo, la auto-documentación y los estándares.

La colaboración del cliente por encima de la negociación del contrato

Tradicionalmente, el usuario o cliente es quien solicita e indica qué debe hacer el software, y espera los resultados, de acuerdo con sus exigencias o expectativas, en los plazos establecidos.

Las metodologías ágiles incluyen de manera directa y comprometida al cliente o usuario en el equipo de trabajo. Es un ingrediente más en el camino hacia el éxito en un proyecto de desarrollo de software. Más que un ambiente de enfrentamiento en el cual las partes buscan su beneficio propio (evadiendo responsabilidades y procurando minimizar sus riesgos), bajo la filosofía de las metodologías ágiles se busca el beneficio común, el del equipo de desarrollo y el del cliente.

La participación del cliente debe ser constante, desde el comienzo hasta la culminación del proyecto, y su interacción con el equipo de desarrollo, de excelente calidad. Es el cliente quien sabe qué es lo que necesita o desea, el más indicado para corregir o hacer recomendaciones en cualquier momento del proyecto.

La respuesta al cambio por encima del seguimiento de un plan

Dada la naturaleza cambiante de la tecnología y la dinámica de la sociedad moderna, un proyecto de desarrollo de software se enfrenta con frecuencia a cambios durante su ejecución.

En este sentido, las metodologías pesadas con frecuencia caen en la idea de tener todo completo y correctamente definido desde el comienzo. No se cuenta entre sus fortalezas la habilidad para responder a los cambios.

Por el contrario, en las metodologías ágiles la planificación no debe ser estricta, puesto que hay muchas variables en juego, y debe ser flexible para poder adaptarse a los cambios que puedan surgir. Una buena estrategia es hacer planificaciones detalladas para unas pocas semanas y planificaciones mucho más abiertas para los siguientes meses.

3.2 Principios

El concepto de *principio* se refiere a la característica que diferencia un modelo tradicional de uno ágil. Se trata de ideas principales a seguir para un desarrollo ágil. Los principios del manifiesto son doce [164]:

1. Nuestra mayor prioridad es **satisfacer al cliente** mediante entregas tempranas y continuas de software con valor.
2. **Aceptamos que los requisitos cambien**, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. **Entregamos software funcional frecuentemente**, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores **trabajamos juntos** de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a **individuos motivados**. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la **conversación cara a cara**.
7. El software funcionando es la **medida principal** de progreso
8. Los procesos ágiles promueven el **desarrollo sostenible**. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La **atención continua** a la excelencia técnica y al buen diseño mejora la agilidad.
10. La **simplicidad** o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de **equipos auto-organizados**.
12. A intervalos regulares el equipo **reflexiona** sobre cómo ser más efectivo para a continuación **ajustar y perfeccionar** su comportamiento en consecuencia.

Las metodologías ágiles dan mayor valor al individuo, a la interacción con el cliente y al desarrollo incremental del software con iteraciones muy cortas.

4 Metodologías ágiles

Las metodologías ágiles están revolucionando la manera de producir software, y han abierto un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales. En la Tabla 5 [167] podemos ver una comparativa entre ambos tipos de metodologías.

Metodologías Ágiles	*Metodologías Tradicionales*
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 5. Desarrollo ágil vs. tradicional

Existen en el mercado varias propuestas sobre cómo implementar los valores y principios enunciados del manifiesto. En ellas, cada autor proporciona su visión personal dependiendo de su experiencia o necesidad. A continuación pasamos a exponer brevemente las más destacadas.

4.1 eXtreme Programming

eXtreme Programming (XP) [165], [173], [174] es una metodología ágil creada por Kent Beck que se centra en la satisfacción del cliente. En vez de esperar una determinada fecha para entregar todo lo posible, este proceso realiza entregas cortas o *iteraciones*, a medida que el cliente las necesita. De este modo, el cambio de requisitos no suponen ningún problema, aun cuando se producen en etapas tardías de desarrollo. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre.

XP favorece el trabajo en equipo; clientes y desarrolladores se integran como iguales dentro de un mismo grupo, y favorecen un buen clima de trabajo en el que se favorece el aprendizaje de los desarrolladores. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico [175].

XP mejora el proceso de desarrollo de software con **cinco valores esenciales**: comunicación, simplicidad, retroalimentación, respeto y esfuerzo. Los programadores tienen una comunicación continua con los clientes y con el resto de desarrolladores manteniendo un diseño simple y limpio. Obtienen retroalimentación al realizar el proceso de pruebas desde primer día y entregan el sistema al cliente tan pronto como sea posible realizando los cambios según se solicitan. Cada pequeño éxito está basado en el respeto a las aportaciones. Con estos cimientos y esfuerzo, los programadores XP son capaces de responder a los cambios de requisitos y tecnología.

El proceso de desarrollo planteado por eXtreme Programming consiste en los siguientes pasos [176]:

1. El cliente define el negocio que quiere modelar con el sistema.
2. El programador estima el esfuerzo de desarrollo.
3. El cliente prioriza qué implementar teniendo en cuenta sus preferencias y el tiempo.
4. El programador construye ese valor de negocio.
5. Vuelta al paso 1.

En todas las iteraciones, programador y cliente aprenden para mejorar. No debemos presionar al programador para realizar más trabajo del previsto puesto que se reflejará en la calidad del resultado negativamente. Del mismo modo, el cliente es el responsable de que el sistema tenga un mayor valor de negocio en cada iteración.

4.2 Scrum

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle [177]. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. La primera es que el desarrollo de software se realiza mediante *iteraciones*, denominadas *sprints*, con una duración de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración. Podemos ver la metodología representada en la Figura 63 [178].

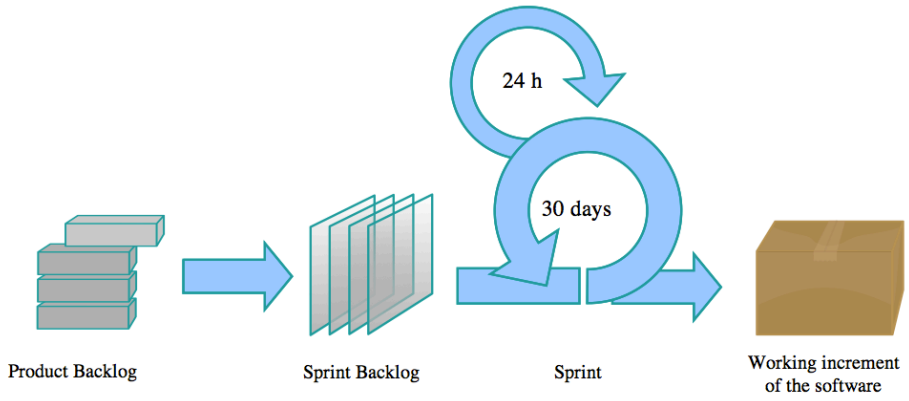


Figura 63. Metodología SCRUM

4.3 Crystal Methodologies

Se trata de un conjunto de metodologías desarrolladas por Alistair Cockburn [179]. Crystal está orientado principalmente a las personas que participan en el proyecto de desarrollo de software, ya que Cockburn entiende que en ellas está la clave para que el proyecto tenga oportunidades de éxito. Según su autor, Crystal está centrado en:

1. Las personas
2. La Interacción

3. La comunidad
4. Las habilidades
5. Los talentos
6. Las comunicaciones

El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25-30 personas)

4.4 Dynamic Systems Development Method (DSDM)

Desarrollada por Stapleton [180], define el marco para desarrollar un proceso de producción de software. Sus principales características es ser un proceso iterativo e incremental con una equipo de desarrollo que trabaja junto al cliente. Propone cinco fases:

1. Estudio de viabilidad,
2. Estudio de la empresa
3. Iteración del modelo funcional
4. Diseño e iteración de la estructura
5. Implementación.

Las tres últimas son iterativas, además de existir realimentación a todas las fases.

4.5 Adaptive Software Development (ASD)

Su impulsor es Jim Highsmith [181]. Se trata de un proceso iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El desarrollo adaptable de software utiliza un ciclo de desarrollo dinámico e iterativo conocido como *Especular-Colaborar-Aprender*. Este ciclo está dedicado a un constante aprendizaje y a una intensa colaboración entre desarrolladores y clientes, esto debido al constante cambio en el ambiente de los negocios.

A diferencia de la mayoría de metodologías de desarrollo de software las cuales utilizan un ciclo de vida estático: *Planear-Diseñar-Construir*; ASD ofrece un ciclo de vida iterativo no lineal, donde cada ciclo puede iterar y ser modificado al tiempo que otro lo hace.

4.6 Feature-Driven Development (FDD)

Sus diseñadores son Jeff De Luca y Peter Coad [182]. FDD está pensado para proyectos cuyo tiempo de desarrollo es relativamente corto (menos de un año). Se basa en un proceso con iteraciones cortas (de unas dos semanas) que producen software funcional que el cliente y la dirección de empresa pueden monitorizar.

Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Un proceso FDD consta de 5 pasos:

1. Desarrollo de un modelo general.
2. Construcción de la lista de funcionalidades.
3. Plan de entregas en función de las funcionalidades a implementar.
4. Diseñar en base a las funcionalidades.
5. Implementar en base alas funcionalidades.

5 Lean Software Development

En su traducción literal, *lean* significa “delgado, flaco” pero fundamentalmente hace referencia al método de producción *lean manufacturing* (LM) adoptado por Toyota en la década de los 50, época en que la fábrica japonesa estaba luchando por recuperarse de la segunda guerra mundial; por ello, es frecuentemente llamado TPS (*Toyota Production System)*. Taiichi Ohno es considerado el padre de este modelo, aunque también fue ayudado por otros compañeros como Shingo o Deming [183].

LM se centra en reducir el gasto innecesario de tiempo y horas/hombre durante el proceso de fabricación y maximizar la calidad del producto. Para acortarlo, LM proporciona un modelo de producción basado en proceso que rápidamente genera productos de calidad exactamente como los quiere el cliente y cuando los quiere el cliente.

El movimiento ágil visto anteriormente se basa en un manifiesto con una filosofía semejante al LM. *Mary* y *Tom Poppendieck* [168] se dieron cuenta de ello y acuñaron el término *Lean Software Development* (LSD) en su libro homónimo. Supone una aplicación al desarrollo de software del proceso industrial de desarrollo LM. Uno de los grandes aciertos del libro de los Poppendieck consiste en no trasladar tal cual el TPS al campo del software, sino abstraer los principios básicos del mismo y trasladarlos teniendo en cuenta que no se trata de un proceso de manufactura sino de desarrollo. Por ejemplo, el desarrollo secuencial puede ser necesario en una cadena de montaje donde los componentes tienen que montarse en un orden determinado, pero no lo es tanto en el desarrollo de software donde los distintos requisitos pueden trabajarse fácilmente en paralelo.

5.1 Principios de LSD

Aunque fue *Robert Charette* el que inicialmente definió los principios del LSD [184], han sido Tom y Mary Poppendieck los que más han contribuido a la difusión de la aplicación de la filosofía lean al desarrollo de software. Los Poppendieck establecen siete principios que analizamos a continuación [185].

1. Eliminar el desperdicio – gasto innecesario (*eliminate waste*).
2. Amplificar el aprendizaje (*amplify learning*).
3. Decidir lo más tarde posible (*decide as late as possible*).
4. Entregar lo antes posible (*deliver as fast as posible*).
5. Dar poder al equipo (*empower the team*).
6. Construir Integridad Intrínseca (*build integrity in*).
7. Tener una visión global (*see the whole*).

5.1.1 Eliminar el *desperdicio*

Desperdicio es todo aquello que no añade valor a un producto, valor percibido por el cliente. En el pensamiento lean, el concepto de desperdicio es un gran obstáculo. Por ejemplo, si un componente está en una estantería almacenando polvo, es desperdicio. Aplicado al desarrollo, si tenemos un libro de requerimientos software acumulando polvo, es un desperdicio. Si una planta de producción genera más unidades de a las que

inmediatamente se les puede dar salida, entonces son un desperdicio. Si los programadores codifican más funcionalidades de las necesarias, entonces, éstas son un desperdicio. Lo ideal es saber qué es lo que el cliente quiere, y entonces fabricarlo o desarrollarlo tal y como lo quiere, de forma inmediata. Todo lo que entorpezca el hecho de tener al cliente rápidamente satisfecho, es un desperdicio.

5.1.2 Amplificar el aprendizaje

El desarrollo es un ejercicio de descubrimiento, mientras que la producción es un ejercicio en la reducción de la variación, y por esta razón, en un enfoque lean presenta técnicas diferentes a las de LM. El desarrollo de software es concebido como un proceso de aprendizaje con el reto añadido de que los equipos de desarrollo son grandes y los resultados son complejos. El mejor enfoque para la mejora de un entorno de desarrollo de software es amplificar el aprendizaje.

5.1.3 Decidir lo más tarde posible

Las técnicas de desarrollo para la toma tardía de decisiones son eficaces en los ámbitos que involucran la incertidumbre. En un entorno de incertidumbre, la mayoría de los mercados económicos ofrecen diversas opciones y de ese modo, evitan que los inversores estén atascados y esperen hasta que el futuro se aproxime y sea más fácil de predecir.

Retrasar las decisiones es valioso porque se pueden tomar mejor cuando estas se basan en hechos, y no en la especulación. En un mercado en constante evolución, mantener las opciones de diseño abiertas es mejor que preestablecerlas. Una estrategia clave para retrasar los compromisos en el desarrollo de un sistema complejo es integrar la capacidad de cambio en el sistema.

5.1.4 Entregar lo antes posible

Hasta hace poco, el desarrollo rápido de software (*Rapid Application Development* [186]), no había sido valorado. Era mejor tomar un enfoque cuidadoso de no-cometer-errores. Sin embargo, el desarrollo rápido tiene muchas ventajas. Sin velocidad, no se pueden retrasar las decisiones. Sin velocidad, no se tiene información fiable procedente del testeo. En desarrollo, el descubrimiento es fundamental para el aprendizaje ya que el

diseño, implementación y retroalimentación, mejoran. La rapidez asegura que los clientes obtengan lo que necesitan ahora, no lo que necesitaban ayer.

5.1.5 Dar poder al equipo

Nadie entiende mejor los detalles de un proyecto que las personas que realmente hacen el trabajo. La participación de los desarrolladores en las decisiones técnicas es fundamental para lograr la excelencia. La gente de producción combina entre todos un gran conocimiento del más mínimo detalle. Un equipo experto de desarrollo, guiado por un buen líder, tomará las mejores decisiones técnicas y de proceso que se puedan tomar.

Debido a que las decisiones se toman tarde y la ejecución es rápida, la figura de un director que organice las actividades de los trabajadores no es viable. El enfoque lean usa técnicas *pull* para programar el trabajo, así como mecanismos de notificación para que unos trabajadores puedan permitir a otros saber qué se necesita hacer. El mecanismo pull consisten en ofrecer versiones cada vez más refinadas del software de trabajo a intervalos regulares. La notificación se produce a través de informes visibles, reuniones diarias, integración frecuente y pruebas exhaustivas.

5.1.6 Construir Integridad Intrínseca

Se percibe que un sistema tiene integridad cuando el usuario piensa*, "¡Sí! Eso es exactamente lo que quiero. ¡Me habéis leído la mente!"*. El software necesita un nivel adicional de integridad ya que debe mantener su utilidad a lo largo del tiempo. Un software con integridad tiene una arquitectura coherente, una alta usabilidad, y además es sostenible, adaptable y extensible.

5.1.7 Tener una visión global

Los sistemas complejos requieren una amplia experiencia en diversas áreas. Uno de los problemas más difíciles con el desarrollo de productos es que los expertos tienen una tendencia a maximizar el rendimiento de la parte que representa su propia especialidad en lugar de centrarse en el rendimiento general del sistema.

Cuando los individuos se centran en su contribución especializada en lugar de rendimiento global surge este problema que se conoce como *suboptimización***.**

6 Agilidad y AMTs

ASD (Agile Software Development) es una filosofía que se separa radicalmente de los métodos tradicionales de desarrollo de software como el modelo de desarrollo en cascada (*Waterfall model*) [187]. Centradas más en la organización de los equipos, las metodologías ágiles como Scrum o eXtreme Programming prestan poca atención a las arquitecturas software [188]. Dar soporte a los equipos con ciertos activos software validados como frameworks de desarrollo o diseños arquitectónicos que fomentan la reutilización y hacen la programación más fácil parecen ser olvidados de momento.

El concepto de arquitectura aterroriza a los equipos de desarrollo ágil que ven en la arquitectura un signo "maligno" del pasado; un mal hábito que solo acarrea toneladas de documentación, diseño inicial (BUFD, *Big Up-front Design*) y aspectos innecesarios (YAGNI, *You Ain't Gonna Need It*) [18]. Los *agilistas* consideran que la arquitectura es algo que debe emerger gradualmente en cada iteración; afrontan los cambios y prefieren sistemas adaptativos en vez de usar arquitecturas predefinidas que limiten la evolución de los sistemas.

Sin embargo, además de estas reivindicaciones, los métodos ágiles se esfuerzan por conseguir entregas funcionales de software en etapas tempranas y de manera frecuente. Si esto es así, puede entonces que merezca la pena empezar desde arquitecturas conocidas y que den soporte en vez de empezar desde cero. En los sistemas cloud MT otros elementos como el rápido despliegue pueden complementar y dar soporte a la filosofía ágil, de forma que en un corto periodo de tiempo se puedan alojar en el sistema a tantos tenants como sea posible.

La arquitectura no debe de ser condenada a una mala praxis contraria a la agilidad, sino como herramientas de soporte y gran ayuda.

Recientemente, destacados académicos del campo de sistemas y de la ingeniería del software reclaman y abogan por la coexistencia entre la agilidad y arquitectura [19]–[21]. Según [18] "*ciertas clases de sistemas que ignoran conceptos arquitectónicos se encuentran sin salida y se derrumban debido a la falta de enfoque arquitectónico"*. En [20], Madison habla de *Arquitectura Ágil* como una combinación de estas dos

vertientes; el arquitecto juega un papel esencial y la nueva dirección arquitectónica debe incluir una gran variedad de opciones en vez de suponer una solución cerrada.

Las arquitecturas MT y otras tecnologías como *Software Product Line Engineering* (SPLE) [189] pueden dar soporte a la agilidad en otras cuestiones complementarias como el rápido despliegue, la calidad o el time-to-market. En [190] la agilidad se define como "*la habilidad para adaptarse al mercado y a aspectos del entorno de forma rápida y económica*". Esta afirmación originalmente aplicada a SPLE puede ser perfectamente extendida a las aplicaciones SaaS MT. En SPLE las modificaciones se realizan sobre los artefactos para adaptarse a los nuevos requerimientos del mercado; estos artefactos son usados en la plataforma de forma que los nuevos productos satisfagan las nuevas necesidades. En multi-tenancy, con tan sólo una instancia de aplicación (excepto multi-tenancy farm), la propagación del cambio es casi instantánea. Es más, los cambios se realizan sobre proyectos existentes y no desde cero.

Las arquitecturas SaaS multi-tenant persiguen las economías en escala debido a la compartición de la instancia de la aplicación entre los tenants, independientemente del tipo de la aplicación desplegada. Los framework administrativos permiten la creación rápida de cuentas de cliente que son capaces de acceder y comenzar a utilizar el sistema en cuestión de minutos (Figura 64). Estas dos opciones no son incompatibles, sino distintos y pueden ser combinadas.

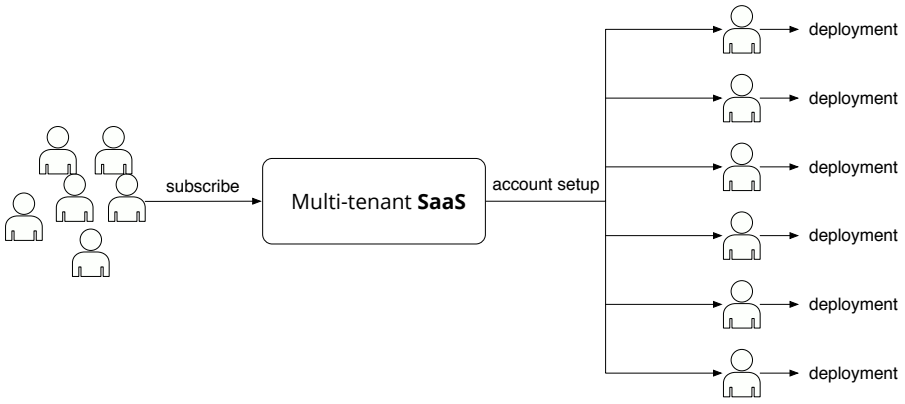


Figura 64. Los sistemas SaaS MT permiten una rápida configuración y despliegue de nuevos tenants

La literatura **todavía no ha profundizado** en cómo esta arquitectura puede ayudar o incluso ser compatible con la agilidad. Sin embargo, las AMTs posibilitan no sólo una

reducción de costes a los clientes y un incremento en el beneficio de los proveedores sino que también complementan y dan soporte a la agilidad en diferentes aspectos:

- **Despliegue**, los clientes son rápidamente registrados en el sistema.
- **Mantenimiento y escalabilidad,** al compartirse la instancia.

Las arquitecturas que den soporte a la agilidad son una pieza clave en este nuevo paradigma SaaS. Sin la rapidez y eficiencia en la gestión de clientes o la sencillez en las actualizaciones que proporcionan las MTAs, SaaS estaría abocado al desastre como su predecesor ASP.

Capítulo VI

Multi-tenancy Multi-target (MT^2)

Se presenta la propuesta consistente en una extensión para arquitecturas multi-tenant. Este novel enfoque persigue mejorar los sistemas multi-tenant tradicionales, de forma que las soluciones resultantes puedan ofrecer varios servicios (ERP, CRM, SCM, etc.) con una única instancia de aplicación. Los proveedores pasan a tener un mercado multi-objetivo (*multi-target*) más amplio, los clientes se benefician de una nueva reducción de precios y los programadores se favorecen de un soporte a la agilidad en el desarrollo mediante la reutilización de componentes arquitectónicos.

1 Introducción

Tal y como hemos visto en el marco teórico, la eficiencia multi-tenant dota a las aplicaciones SaaS de un alto nivel de escalabilidad permitiendo a varios clientes compartir la misma instancia de la aplicación. A nivel de arquitecturas software, multi-tenancy posibilita que varios clientes ejecuten un mismo código, pero pudiendo personalizar la solución software a varios niveles por medio de los metadatos. Esta situación ha permitido un abaratamiento de costes y por tanto, un éxito de la nueva fórmula de comercialización. Sin embargo, se han detectado inconvenientes o mejoras que se podrían aplicar al modelo tradicional.

Tradicionalmente, las aplicaciones MT son compartidas entre tenants pertenecientes a un mismo sector o con las mismas necesidades funcionales. Un CRM será usado por aquellas empresas que necesiten tener un mejor control sobre las relaciones con sus clientes, mientras que un SCM por empresas con canales de suministro complejos y que necesiten ser competitivas. Según esto, podemos decir que las aplicaciones actuales despliegan una única funcionalidad y que por tanto, los vendedores de SaaS MT alojan en sus servidores a tenants pertenecientes a un solo sector del mercado (*target*). Además, los desarrolladores deben de programar e implantar una aplicación por cada software, es decir por funcionalidad, suponiendo un coste económico y un aumento del time-to-market.

Sin embargo, existen muchos componentes de la arquitectura que podrían ser compartidos entre aplicaciones de diferente natura. Por ejemplo, cuando un desarrollador programa la autenticación en el sistema, le da igual si este es un ERP o un CRM; simplemente busca un acceso correcto y seguro de los usuarios finales a la aplicación. Esta *duplicidad de componentes* ocurre no sólo a nivel de código fuente, sino que también a nivel de presentación como son los elementos de interfaz (iconos, gráficos, hojas CSS de estilo, etc.).

En definitiva, existen una serie elementos que son copiados de forma idéntica en las aplicaciones MT actuales. Si cloud computing y SaaS tienen como base principal y clave el uso compartido **¿por qué no compartir también estos componentes?**

Basado en la reutilización de componentes comunes, nuestro enfoque multi-tenancy multi-target (MT^2) **extiende las arquitecturas MT tradicionales para que** las aplicaciones puedan no sólo alojar varios clientes, sino también **varias funcionalidades**. Esta selección de servicios es desplegada dinámicamente y de forma selectiva en función del contrato de suscripción. Los elementos comunes entre aplicaciones pasan a ser **componentes reutilizables** entre funcionalidades, dejando de ser duplicados.

Gracias a MT^2, una sola aplicación será necesaria para dar servicio a clientes con necesidades funcionales múltiples como un CRM, ERP o SCM (entre otras). Este entorno multi-funcional supone **nuevos beneficios** para todos los actores de cloud computing. Las empresas ven reducidos de nuevo los precios, los vendedores amplían su rango comercial a un mercado multi-target y los programadores abrazan este modelo por su soporte a la agilidad fomentando la reutilización.

De este modo, MT^2 extiende la arquitectura MT para obtener aplicaciones con aún mayores beneficios de los que ya proporciona. Además, estos son aplicables no sólo a los clientes, sino que también a los vendedores y los desarrolladores de software.

Por sencillez en la exposición, en este trabajo hemos fijado la *granularidad funcional* a un nivel alto, identificando una funcionalidad con aplicaciones completas como ERP, CRM o SCM. Sin embargo, en nuestro enfoque se puede reducir el concepto y entender como funcionalidad una necesidad de menor complejidad; por ejemplo, un sistema de control de horarios, una agenda compartida, calendario de citas, etc.

La exposición del enfoque intenta ser consistente con la línea seguida en el desarrollo del marco teórico. En primer lugar, recordamos las arquitecturas MT tradicionales y los inconvenientes detectados en las mismas. Posteriormente, damos paso a presentar nuestra propuesta; exponemos sus fundamentos y explicamos cómo esta novel extensión resuelve dichos inconvenientes brindando nuevas ventajas a la comunidad del desarrollo de software. Finalmente, las siguientes secciones estudian los beneficios del enfoque en términos de agilidad y sus aportaciones al mundo del software empresarial, especialmente para las pequeñas y medianas empresas.

2 Multi-tenancy *Mono-target*

El mercado de los sistemas de información (EIS, *Enterprise Information Systems*) es muy extenso y los ejemplos más típicos de funcionalidades ofrecidas como servicio son CRM, ERP o SCM. Existen muchas empresas que ya comercializan estas funcionalidades como SaaS y el abanico de ofertas no es sólo muy amplio, sino también dispar. Hoy en día podemos suscribirnos a CRM horizontales como el conocido *Salesforce.com* [191] o bien soluciones no tan célebres, pero específicas de industria (verticales [87]) como *NetPropertyAgent.com* [138]. En otro ámbito de necesidades funcionales también podemos encontrar un CMS cloud [192] o incluso un servicio para la gestión de eventos y congresos [193].

En MT tradicional**,** los suscriptores obtienen una misma funcionalidad personalizada por la capa de metadatos de la arquitectura, pero siempre dentro de una misma línea de negocio (**LOB**, *Line Of Business*) [16], [66]. Para una empresa, encontrar un único software MT que cubra todas sus necesidades funcionales es una **tarea ardua, por no decir imposible**. Por ejemplo, si un tenant que está suscrito a una aplicación que despliega CRM como funcionalidad y necesita CMS, probablemente tenga que suscribirse a una nueva aplicación de un proveedor diferente. Por regla general, diferentes funcionalidades implican diferentes aplicaciones.

Los **proveedores** por su parte, tienen el mercado reducido a la funcionalidad desplegada por su aplicación. El proveedor de un servicio ERP tendrá como mercado potencial aquellas empresas además presenten necesidades de gestión de recursos empresariales (y que además lo puedan pagar). Análogo al ejemplo anterior, un proveedor de ERP nunca podrá optar con la misma aplicación al intervalo de mercado con necesidades funcionales de CMS y viceversa.

En este sentido, podemos decir que **las soluciones multi-tenant actuales son *mono-target*** ya que sólo desarrollan una funcionalidad y los proveedores ven restringida su cuota de mercado a un único nicho comercial, definido por la funcionalidad desplegada.

Desde el punto de vista del **programador**, el entorno mono-funcional de las aplicaciones también puede ser mejorado. Multi-tenancy ha conseguido rebajar los esfuerzos de mantenimiento y desarrollo al unificar las instancias de las aplicaciones en

una compartida. Sin embargo, **a nivel funcional multi-tenancy sigue siendo multi-instancia**; los desarrolladores necesitan implantar instancias diferentes para funcionalidades distintas.

2.1 Inconvenientes mono-target

El multi-tenancy tradicional presenta por tanto una serie de contrariedades debido a su naturaleza mono-funcional que afectan a usuarios, vendedores y programadores. A continuación, resumimos por cada actor en cloud computing los inconvenientes detectados ilustrándolos con ejemplos de diversos escenarios.

2.1.1 Clientes

En la Figura 65 planteamos un escenario de tres funcionalidades (ERP, CRM Y SCM); cada una de ellas es servida por un proveedor distinto y el tenant necesita desplegar en su organización todas ellas. En la situación actual, dado que cada proveedor es capaz de dar servicio a una sola funcionalidad, el cliente debe establecer contratos con cada uno de ellos, equiparando así las necesidades funcionales al el número de aplicaciones, vendedores y cuotas.

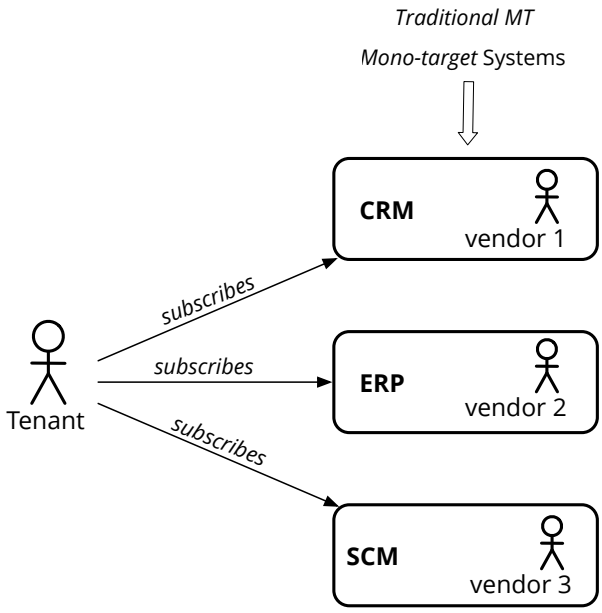


Figura 65. MT tradicional (Mono-target). El cliente debe contratar una aplicación por servicio

Los **problemas** detectados desde el punto de vista del cliente los podemos aunar en la siguiente lista:

- Aumento del precio de suscripción al tener que contratar tantas aplicaciones como funcionalidades.

- Incremento de costes de mantenimiento y soporte.

- Dificultad en aprendizaje, los tenants tienen que adaptarse a la interfaz y a las características particulares de cada aplicación SaaS contratada.

- Mayor complejidad operacional, al tener que tratar con tres empresas proveedoras diferentes.

- Mala escalabilidad funcional; cambios en los requisitos funcionales implican desarrollos a medida o nuevas suscripciones.

- El escollo anterior también está relacionado con las dificultades en la migración o de datos o integración de sistemas. El cambio de aplicación SaaS o nuevas contrataciones implican la migración de información o integración de sistemas con la complejidad que ambos procesos significan.

En una **situación ideal**, tendríamos **un solo proveedor** que ofreciera un **único software** económico, robusto, eficiente y con escalabilidad funcional que cubriera todas sus necesidades funcionales.

2.1.2 Proveedores

El target o cliente objetivo de un vendedor de software está estrechamente relacionado con las características y funcionalidades que implementa su aplicación. Aunque según [131] existen solapamientos funcionales en los EIS, los sistemas cumplen con requisitos funcionales a un nivel de complejidad diferente, que las distingue unas de otras. Por ello, las empresas **deben comercializar diferentes soluciones** para cubrir necesidades funcionales o de complejidad desiguales. Como ejemplo de esta oferta múltiple , Oracle ofrece actualmente en su página web 23 soluciones empresariales [194].

La Figura 66 se centra ahora en proveedores, concretamente en el vendedor 2. Su aplicación despliega ERP y por tanto, solamente puede acceder al mercado de las empresas con esas necesidades funcionales. Si el sistema ofreciera también CRM, entonces tendría también como cliente potencial al Tenant 1; además no solo eliminaría

la posibilidad de perder al Tenant 2 (por migración de aplicación) sino que podría incluso aumentar su suscripción por incrementar los servicios proporcionados (sumaría CRM a la funcionalidad ERP existente).

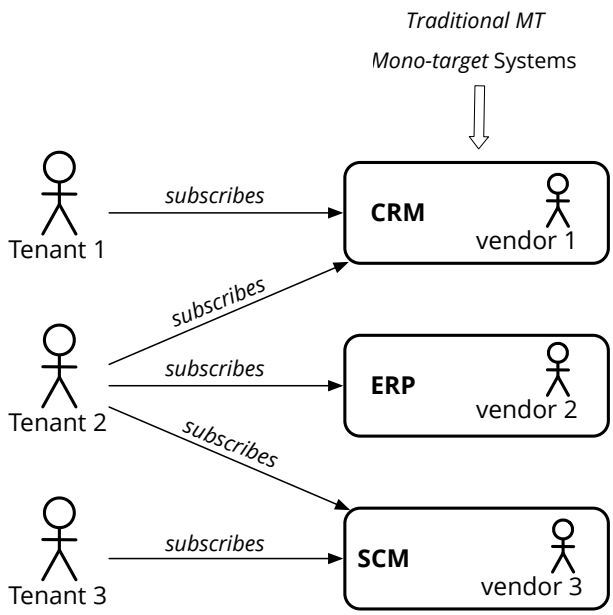


Figura 66. El vendedor de software sólo opta a nicho de mercado

Multi-tenancy tradicional divide el mercado potencial en **grupos disjuntos**. Las empresas que comercialicen a un determinado grupo, tienen que desarrollar nuevas aplicaciones si quieren vender a otro nicho diferente.

Podemos reducir la granularidad funcional y trasladar los mismos ejemplos a situaciones de necesidades funcionales menores y específicas. Por ejemplo, un vendedor especializado en un software de control de presencia, no podrá competir por la cuota de mercado con otro especializado en monitorización de posicionamiento en redes sociales. En cualquier caso, independientemente del nivel de granularidad funcional no hablamos de competir por liderazgo en una cuota o nicho de mercado, sino **al menos, tener acceso a ella.**

Por todo lo anterior, es difícil encontrar (sino imposible) proveedores de EIS en modalidad SaaS pertenecientes al rango de la pequeña y mediana empresa. El coste de desarrollo, las necesidades de infraestructura y personal así como el riesgo que corren de pérdida de clientes (por migraciones) hacen que este modelo sea inviable para este intervalo pyme.

Podemos resumir los **inconvenientes** de los proveedores en:

- División del clientes en grupos disjuntos. Reducción del mercado potencial a un solo nicho relacionado con la funcionalidad desplegada.
- La expansión del negocio a otras funcionalidades es compleja. Implica la creación de nuevas aplicaciones y aumento de costes de desarrollo, mantenimiento e infraestructura.
- Riesgo de pérdida de clientes por falta de despliegue funcional en la aplicación.
- Cuota de reservada a las grandes empresas por los elevados costes y riesgos de comercialización.

2.1.3 Programadores

En el proceso de desarrollo de software, existen **componentes comunes e independientes al tipo de aplicación**. Elementos como librerías de consulta a la base de datos o de acceso al sistema de ficheros son idénticos y usados en aplicaciones distintas. Sin embargo y a pesar de ser los mismos componentes, los programadores los duplican, instalan y reconfiguran en cada EIS.

Por ejemplo, en un nivel de granularidad pequeña, una función que nos transforme el formato de fecha americano (*Año-Mes-Día)* al formato Europeo (*Día-Mes-Año*), podremos definirla como una función horizontal, con un propósito general y válida para cualquier aplicación e independiente de la línea de negocio. Otro ejemplo claro de horizontalidad es la autenticación para entrar al sistema, que típicamente consiste en cotejar el nombre de usuario y la contraseña para conceder permisos de acceso al sistema.

En la situación actual, existen elementos arquitectónicos que pueden ser compartidos y reutilizados entre aplicaciones de diferente naturaleza, y son sin embargo calcados en cada solución, consumiendo esfuerzos y aumentado los tiempos de entrega.

No estamos descubriendo nada nuevo, la *reusabilidad* y la *encapsulación* son aspectos fundamentales en la programación. Todos los desarrolladores hacen uso de librerías ya implementadas y testadas que son copiadas en cada software que requiera hacer uso de

estas funcionalidades. El mismo tipo o incluso **la misma función exacta**, la encontramos en un CRM o un ERP de servicios SaaS distintos. Sin embargo, en ambientes MT mono-target **cada implementación implica una replicación de estos componentes** con el consiguiente aumento de tiempo, esfuerzo y costes, que finalmente se verán trasladados a los clientes.

Este hecho es ilustrado en la Figura 67 en donde los componentes horizontales comunes has sido recalcados en color gris oscuro; en ella podemos ver cómo el programador tiene que duplicarlos en todas las implantaciones MT tradicionales del escenario. Destacar que dentro de estos elementos comunes también están incluidos los *Agentes de Integración Cloud* (CIA, Cloud Integration Agents) que estudiábamos en la sección de arquitecturas multi-tenant y cuyo objetivo es la integración de sistemas.

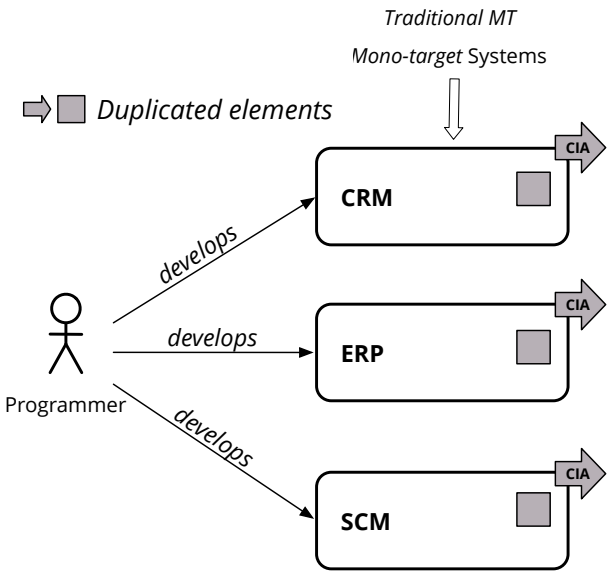


Figura 67.MT tradicional: El programador debe copiar componentes en cada aplicación

Tal y como hemos hecho para los clientes y proveedores, enumeramos un resumen de **inconvenientes** detectados en el lado de los programadores:

- Aumento del esfuerzo de implantación y desarrollo, aumentando por consiguiente los tiempos de entrega, el coste y el time-to-market.
- Mantenimiento y soporte complejo. Por cada funcionalidad se tiene que atender a una aplicación diferente.

- Incremento de la complejidad en la integración, al tener que crear múltiples pasarelas**.**

3 Multi-tenancy Multi-target

Podemos definir *Multi-tenancy Multi-target* (MT^2) como una **extensión para arquitecturas MT** que permite que un mismo sistema operacional pueda alojar y desplegar bajo demanda múltiples funcionalidades. Estas funcionalidades son servidas de forma selectiva, dependiendo del contrato de suscripción elegido por cada cliente. De este modo, organizaciones de diferente industria, con necesidades funcionales dispares o simplemente con niveles de complejidad desiguales pueden ser alojados en una misma solución software. Las aplicaciones MT tradicionales dejan de ser mono-target (para un solo tipo de cliente) y se convierten en multi-target (varios tipos de clientes).

En MT^2 las aplicaciones mono-target **se integran como funcionalidades** dentro del mismo sistema. Sin embargo, no todas son desplegadas ya que su ejecución se controla de forma selectiva dependiendo del contrato. El conjunto de funcionalidades que un sistema MT^2 puede desplegar se conoce con el nombre de ***portfolio funcional***. El número de funcionalidades presentes en el portfolio puede diferir de unos sistemas a otros y depende de la aplicación.

MT^2 es **escalable** no sólo a nivel de tenants, sino también a nivel funcional. Los sistemas MT^2 puede que desplieguen pocas funcionalidades al principio y ofrezcan un portfolio escaso, pero pueden aumentarlo con el tiempo. En teoría, los sistemas MT^2 antiguos tienen un portfolio mayor, ya que los clientes solicitan el desarrollo de nuevas funcionalidades y una vez terminadas, éstas son incorporadas en el sistema pudiendo ser ofrecidas al resto de los tenants presentes y venideros, hasta que estén obsoletas.

Los tenants disponen de un catálogo del que escoger configurando su ***suscripción funcional***. Esta información es almacenada en el nivel administrativo y supone un nuevo nivel de personalización de los tenants, que se suma a los niveles proporcionados por los metadatos en los sistemas MT tradicionales. Un nuevo tipo de ***metadatos multi-target*** (*metadatos* MT^2) precisa registrar el contrato de los clientes almacenando las funcionalidades que desplegar a nivel de instancia para ese tenant concreto.

Con MT^2, los proveedores pueden expandirse más fácilmente a otros LOBs, ya que corren menos riesgos por la naturaleza escalable funcional de los sistemas. Por tanto, las empresas **aumentan el rango de clientes potenciales y reducen los costes** generales de mantenimiento, infraestructura y comercialización. Por consiguiente, el precio es también rebajado a los clientes, que además disminuyen la complejidad de aprendizaje al unificar las aplicaciones en un única suscripción. Además, nuevos requisitos pueden ser incorporados como funcionalidades en el portfolio, evitando migraciones y costes extra por nuevas contrataciones.

Multi-target añade a multi-tenancy una reutilización de componentes que brinda a los programadores un soporte a la agilidad debido a la eliminación de duplicaciones innecesarias. La principal idea tras MT^2 es la *reusabilidad*; componentes comunes en la arquitectura MT^2 son reutilizados entre funcionalidades y dejan de ser duplicados. En MT^2 existe una sola instancia de aplicación (salvo *multi-tenancy farm*), que engloba varias funcionalidades y las despliega según el contrato de suscripción de los clientes.

En resumen, la arquitectura que proponemos es multi-tenant y multi-target, lo cual proporciona beneficios a todas las partes implicadas. MT^2 reduce considerablemente los precios, el esfuerzo de desarrollo y la complejidad operativa de las aplicaciones SaaS, además de ajustarse mejor a las necesidades funcionales de los tenants.

3.1 Fundamentos MT^2

Los inconvenientes detectados en MT mono-target son causados en gran medida por la duplicidad de componentes idénticos a lo largo de diferentes aplicaciones. La idea principal de este enfoque es aprovechar la reusabilidad y potenciar los sistemas MT tradicionales para que desplieguen no sólo una funcionalidad, sino varias. De este modo, no es necesaria la duplicación ya que las funcionalidades compartirán componentes arquitectónicos de una misma aplicación. Tenants con diferentes necesidades funcionales (o de complejidad) pueden ser alojados en el mismo sistema. Los sistemas MT^2 no tienen por qué desplegar todas las funcionalidades del portfolio y por tanto, deben incluir un nuevo tipo de metadatos para almacenar las funcionalidades a ejecutar por cada tenant.

Arquitectónicamente, el enfoque se basa en la modificación y adicción de componentes para aprovechar la reutilización. En esta extensión, los componentes comunes de las MTA no son duplicados en implantaciones, sino reutilizados por un mismo sistema MT^2 que engloba todas las funcionalidades. Para ello, el **nivel de negocio tradicional se divide en dos**; una capa subyacente conteniendo los elementos horizontales comunes e independientes del LOB y otra constituida por aquellos elementos verticales, específicos de industria.

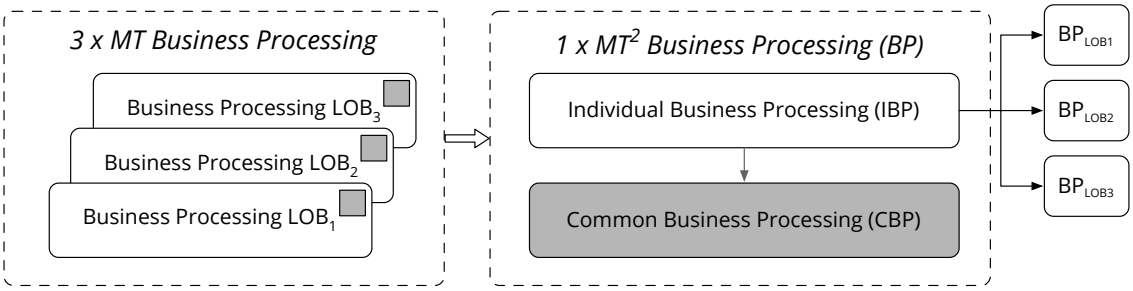


Figura 68. División de nivel de negocio tradicional en MT^2

En la Figura 68 vemos un sistema MT^2 (en la derecha de la imagen) que consolida tres aplicaciones MT tradicionales mediante la creación de una capa de procesamiento común (CBP, *Common Business Processing*) conteniendo los elementos comunes de los LOBs (en gris). Aparte y por encima de ella existe una capa de procesamiento individual (IBP, *Individual Business Processing*) que despliega características verticales, específicas de LOB y no comunes a todas las funcionalidades.

En esta tesis estudiaremos en detalle estos nuevos componentes y analizaremos cómo aprovechar esta división para permitir el despliegue selectivo sin afectar al rendimiento del sistema por sobrecarga.

3.2 Características de MT^2

MT^2 propone una solución a los inconvenientes detectados en MT tradicional e implica mejoras para clientes, proveedores y programadores. Del mismo modo que en la sección anterior, vamos a plantear escenarios MT^2 análogos para ilustrar mejor estos beneficios.

3.2.1 Clientes

En la Figura 69 mostramos el nuevo escenario en el que el vendedor 1 mantiene un sistema MT^2 con un portfolio funcional que ofrece simultáneamente CRM, ERP y SCM. A diferencia del entorno tradicional (Figura 65), el tenant 2 no tiene por qué suscribirse a varias aplicaciones (incrementando complejidad y costes) sino que tan sólo precisa de una suscripción multi-funcional con un único vendedor.

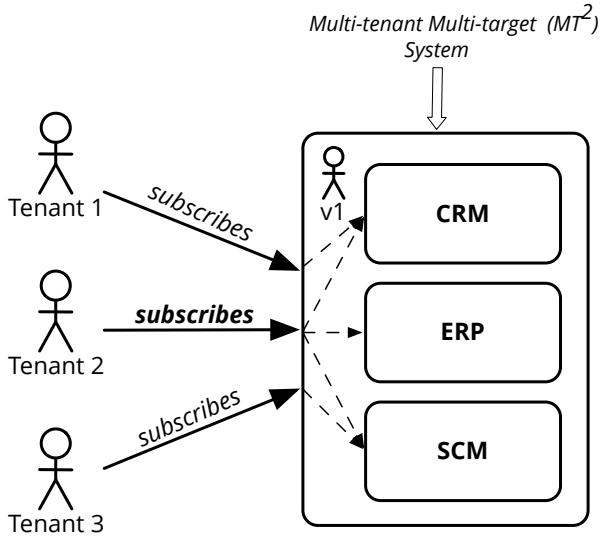


Figura 69. MT^2. Unificación de servicios

Así, los beneficios que aporta MT^2 desde el punto de vista del cliente los podemos resumir en:

- Nueva rebaja de costes al unificar las aplicaciones en una sola; tanto de gastos iniciales (*capex*, capital expenses), así como de costes operativos de mantenimiento y soporte (*opex*, operational expenses).
- Facilidad de aprendizaje, al tener una misma interfaz común que despliega funcionalidades diferentes.
- Tratamiento con un único proveedor, simplificando la operativa de la empresa.
- Nuevos requisitos se traducen en nuevas funcionalidades y no en nuevas aplicaciones. Escalabilidad funcional sencilla.

- El despliegue de funcionalidades no contratadas pero presentes en el portfolio es casi instantáneo. Los clientes no necesitan esperar a la finalización de nuevos desarrollos para comenzar el uso de los mismos.

- Las actualizaciones de seguridad y mejoras sugeridas de otros clientes, aún siendo de funcionalidades distintas se aplican de forma instantánea si estas pertenecen a la capa CBP.

- Desaparece la necesidad de migración o integración de datos con nuevos sistemas, al estar todas las funcionalidades en una misma aplicación.

3.2.2 Proveedores

El vendedor de software amplia su abanico de clientes potenciales ya que empresas pertenecientes a sectores diferentes o con necesidades funcionales distintas pueden suscribirse al mismo software. Por ejemplo, el proveedor de la Figura 70 con un solo sistema tiene acceso a tres nichos de mercado, cuando en mono-target necesitaría tres aplicaciones diferentes.

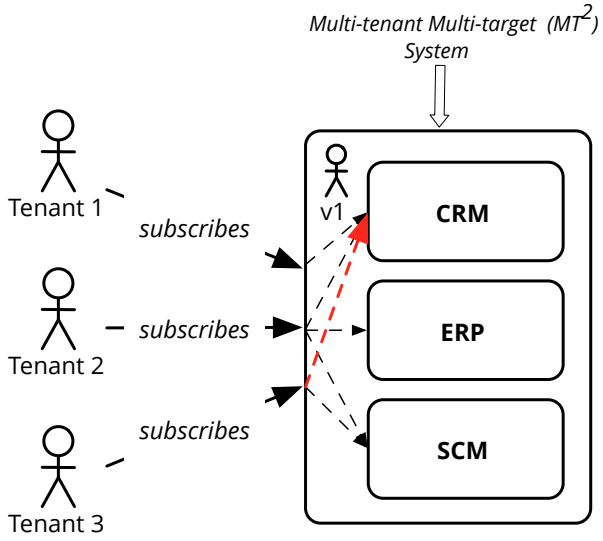


Figura 70. MT² amplia el target a los proveedores y le dota de herramientas de fuerza comercial

Los vendedores tienen también una mayor fuerza comercial ya que pueden "ofertar" funcionalidades no contratadas a los clientes, pero existentes en el portfolio. Así, el Tenant 3 (suscrito sólo a SCM) podría tener una oferta atractiva de contratación de CRM que complementara a su SIE actual. Además, la suscripción sería *instantánea* sin

implantaciones, ya que bastaría con modificar los metadatos MT^2 de su contrato, para relacionarlo también con esta funcionalidad (línea roja en la figura).

Podemos resumir los beneficios de cara a los proveedores SaaS en la siguiente lista:

- Aumento del espectro de clientes potenciales abarcando un mayor número de nichos relacionados con el número de funcionalidades desplegadas.
- Reducción del time-to-market, desarrollos más rápidos y despliegues instantáneos de nuevas contrataciones.
- Simplificación de costes operativos de mantenimiento y de personal por unificación de sistemas.
- Facilidad de expansión a otros sectores software. La escalabilidad funcional y la reutilización de componentes disminuye los riesgos; basta con incorporar la nueva funcionalidad al portfolio.
- Disminución de riesgo de migración de clientes por falta de funcionalidad. Es más, MT^2 implica nuevas oportunidades de venta al poder ofrecer funcionalidades no contratadas a los clientes.
- Oportunidad de negocio para las pequeñas empresas. La disminución de costes y aumento de mercado puede animar a las pymes a la migración del modelo de negocio a SaaS.

3.2.3 Programadores

MT^2 habilita un soporte a la agilidad tanto en el proceso de desarrollo como en el de despliegue. El *desarrollo* se ve simplificado por la reusabilidad de componentes, antes duplicados y reconfigurados en cada sistema. El *despliegue* a su vez también se ve beneficiado puesto que con un sólo nivel administrativo MT^2 podremos desplegar múltiples servicios. El soporte a la agilidad es **una de las grandes aportaciones** del enfoque y será debatido con mayor profundidad en una sección específica.

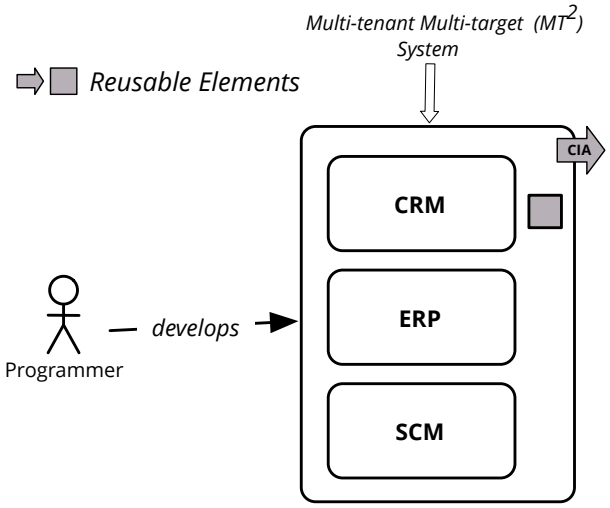


Figura 71. MT²: Agilidad en el Desarrollo y la Implantación

En la Figura 71, los desarrolladores trabajan sobre un único sistema de tres funcionalidades que comparten componentes comunes de la CBP (en gris oscuro) y no tienen que ser duplicados. Como podemos observar, dentro de estos componentes se encuentran también los componentes de integración cloud (CIA), lo cual simplifica los procesos de migración e integración de sistemas. A diferencia del caso mono-target, el equipo tiene que desarrollar un único sistema MT² y no tres MT tradicionales. Además, si un cliente desea la implantación de una funcionalidad no contratada, esta sería desplegada automáticamente.

MT² habilita una reducción drástica del **time-to-market** para el caso de que un cliente solicite una nueva funcionalidad ya que pueden ocurrir dos situaciones:

- **El servicio deseado está presente** en la cartera funcional con lo cual el administrador del sistema MT² tan sólo tendrá que modificar el contrato de suscripción funcional.

- **El servicio deseado no esté presente** por lo que habrá que incorporarlo al portfolio desarrollándolo. En este caso:

    - El tiempo de respuesta sería también menor que en el caso tradicional puesto que como veremos más adelante, existe también un soporte a la agilidad que reduce los tiempos de desarrollo.

- El nuevo servicio, una vez terminado pasaría a formar parte de la cartera funcional y estará disponible de forma inmediata para futuras peticiones.

Existe un tercer caso en el que el servicio deseado esté presente en el portfolio, pero los requisitos varíen en nivel de complejidad. En este escenario, MT^2 también supone beneficios; la empresa puede articular el nivel de complejidad como una nueva funcionalidad reutilizando en este caso un mayor número de componentes. Listamos a continuación el resumen de beneficios de cara al programador:

- Mantenimiento y soporte sencillo de las aplicaciones. Sólo hay que trabajar con un entorno MT^2 y no con múltiples aplicaciones mono-target.
- Soporte a la agilidad en el desarrollo y despliegue reduciendo tiempos de entrega, esfuerzos y time-to-market.
- Facilidad para la integración de sistemas, ya que los CIAs pertenecen a los elementos reutilizados de la arquitectura.

3.3 Comparativa de escenarios

Con el objetivo de facilitar el entendimiento del enfoque y las ventajas que supone con respecto a MT tradicional hemos desarrollado la Figura 72. En ella se compara un escenario mono-target tradicional con un sistema MT^2 y representa una visión conjunta todos los escenarios mostrados hasta ahora.

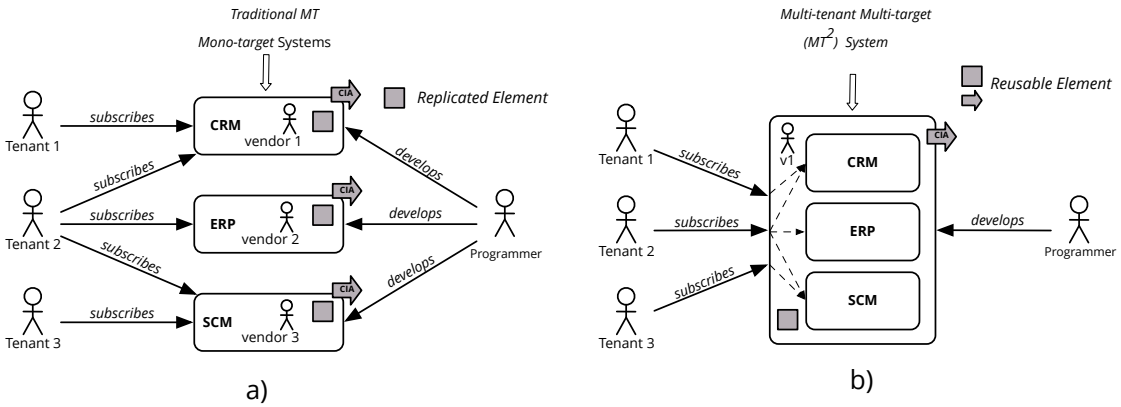


Figura 72. Visión conjunta de ambos enfoques: MT (a) vs MT^2 (b).

En entornos mono-target (Figura 72-a) las aplicaciones son compartidas entre tenants con necesidades funcionales comunes. Debido a este escenario mono-funcional, los

clientes tienen que suscribirse a tantos servicios como funcionalidades (aplicaciones) necesiten. Además, muchos de los componentes de la arquitectura como autenticación, gráficos o componentes CIA son duplicados en las implantaciones mono-target. En la Figura 72-a, el tenant 2 tiene que suscribirse a tres aplicaciones diferentes, con el consiguiente incremento económico y demás inconvenientes detectados.

Tal y como enseña la Figura 72-b, en un escenario MT^2 el tenant 2 unifica sus suscripciones dentro de una única aplicación. La nueva aplicación es un sistema MT^2 que es mantenido por un único proveedor, que además puede ahora acceder ahora a los nichos de mercado de ERP y SCM. Desde el punto de vista del programador, se reduce el esfuerzo y respalda a la agilidad, al convertir los componentes duplicados en elementos reusables entre funcionalidades.

4 MT^2 y Agilidad

En este trabajo hemos analizado el debate actual sobre la coexistencia entre agilidad y arquitectura. Tras años de incompatibilidad, notables académicos reivindican que ambos conceptos no deben ser considerados opuestos, sino compatibles. Las AMTs dan soporte a la entrega rápida de las aplicaciones funcionales y no deben de ser despreciadas sino valoradas como activos en el proceso de desarrollo. En este sentido, las arquitecturas MT tradicionales apoyan la agilidad debido al uso compartido, ya que los despliegues son automáticos.

Sin embargo, el MT tradicional presenta inconvenientes relacionados con su naturaleza mono-target. Y es que, en términos de agilidad **cualquier duplicación implica la pérdida de un tiempo valioso** (y caro). Durante el desarrollo de las aplicaciones, los programadores tienen que duplicar y adaptar/reconectar componentes en las diferentes implementaciones; los proveedores deben dar soporte y mantener varias aplicaciones, teniendo incluso clientes duplicados en los distintos niveles administrativos. Los clientes por su parte sufren un aumento de coste por las múltiples aplicaciones suscritas, además de multiplicar su esfuerzo de aprendizaje teniendo que adaptarse a las particularidades de funcionamiento de las diferentes soluciones software.

MT^2 es un equivalente de reusabilidad (y por tanto de **soporte a la agilidad**). Esto lo consigue eliminando duplicaciones que dotan al sistema de una capacidad de uso compartido no existente en MT tradicional. MT^2 elimina las barreras de agilidad

presentes en el modelo tradicional ya que unifica aplicaciones y simplifica esfuerzos, al poder ofrecer y desplegar varios servicios en vez de uno.

MT^2 es claro ejemplo de alineación agilidad/arquitectura ya que hace posible entregas rápidas de software funcional a los clientes aprovechándose de arquitecturas software consolidadas y probadas. **Los equipos ágiles ven facilitado su trabajo ya que una sola aplicación puede servir para alojar múltiples funcionalidades**.

El enfoque MT^2 respalda la agilidad tanto en el desarrollo de las aplicaciones como el despliegue, en las siguientes subsecciones analizaremos en detalle cada uno de ellos.

4.1 Agilidad en el desarrollo

La reutilización de componentes es la base que ha dado origen a la extensión MT^2. Recordemos que el cambio fundamental suponía la división del nivel de negocio tradicional en dos:

- Capa de procesamiento común (CBP, *Common Business Processing*) conteniendo, componentes horizontales, válidos para todas las funcionalidades.
- Capa de procesamiento individual (IBP, *Individual Business Processing*), comprendida por elementos propios de la funcionalidad, utilizados únicamente por ese servicio del portfolio.

Los elementos CPB son reutilizados entre funcionalidades durante la ejecución de aplicaciones de los tenants. La agilidad en el desarrollo se ve favorecida gracias al proceso de reutilización por dos razones:

1. No es necesario desarrollar componentes ya existentes en los CBP.
2. El desarrollo de los elementos IBP también se ve beneficiado ya que podemos extender el comportamiento de un elemento CBP para particularizar las necesidades propias del proceso de negocio individual (Figura 73).

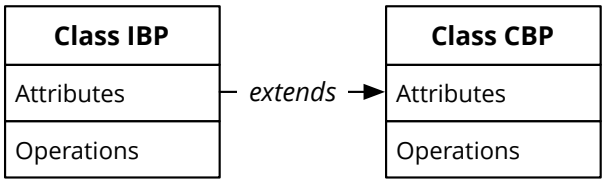


Figura 73. Agilidad en el desarrollo de un componente IBP por extensión de CBP.

Existe un **tercer caso** en el que la clase IBP extienda otra clase IBP. Esta opción es válida siempre que la clase padre sea del mismo componente IBP. Los componentes IBP son importados dinámicamente dependiendo de las funcionalidades contratadas. No debemos nunca extender una clase que puede que no sea cargada en tiempo de ejecución al no estar contratado el servicio.

4.2 Agilidad en el despliegue

Los componentes MT^2 del nivel administrativo dotan al sistema de un soporte a la agilidad en la implantación y de reacción al cambio **superior al modelo tradicional** mono-target. En aplicaciones tradicionales, los clientes pueden ser dados de alta en el servicio de forma rápida y ágil ya que basta con registrar al cliente y establecer las condiciones temporales y de pago de la suscripción. Sin embargo, en este modelo, aplicaciones diferentes suponen implantaciones diferentes y por ende la consiguiente pérdida de tiempo en la instalación y el despliegue.

Con esta extensión, los proveedores SaaS pueden unificar todas sus aplicaciones y configurar un sistema MT^2 que ofrezca en su portfolio todas funcionalidades que previamente desplegaban sus aplicaciones MT tradicionales. Esta centralización mejora la agilidad en el despliegue; con un solo nivel administrativo MT^2 podemos controlar a todos los clientes y funcionalidades (antiguas aplicaciones).

Ya no son necesarias múltiples implantaciones, sino que **un solo sistema MT^2 basta para desplegar todas las necesidades funcionales de los distintos clientes**. A nivel de arquitectura, proporcionamos a los equipos ágiles de herramientas que facilitan su trabajo y reducen los tiempo de respuesta.

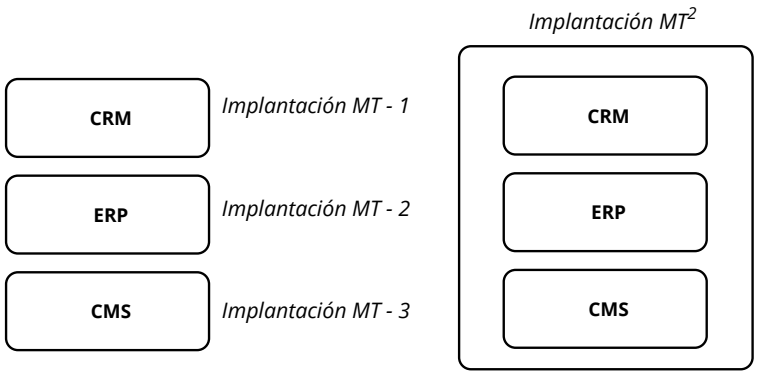


Figura 74. MT vs MT². Agilidad en el despliegue

En la Figura 74 vemos un ejemplo de reducción claro de tiempos de implantación enfrentando los escenario MT y MT². En el primer caso, ante una necesidad de tres funcionalidades tenemos que realizar equivalentes implantaciones. En MT² sólo es necesaria siempre una implantación.

MT² alcanza un escalón superior en agilidad en la implantación, ya que permite una versatilidad funcional que no lo tiene el modelo tradicional. Los suscriptores no sólo disponen de un servicio en poco tiempo, sino que también de un amplio abanico de posibilidades funcionales no presente en el modelo mono-target.

5 MT² y la pequeña empresa

Multi-tenancy es sustancial para el éxito de SaaS ya que permite una reducción de precios que hace posible que las empresas puedan permitirse el acceso a aplicaciones de calidad y de forma casi instantánea. Además, las organizaciones pueden personalizar las soluciones y adaptarlas a sus necesidades mediante el uso de metadatos. Esta fórmula es especialmente atractiva para las pymes debido a que eliminan los costes de inversión inicial (*capex*, capital expenses) y además los operacionales (*opex*, operational expenses) se ven reducidos de forma drástica.

Estudiamos en el marco teórico como el rango en el cual están encuadradas las pymes es muy amplio (hasta 500 en el caso de EEUU, 250 para la UE). Sin embargo, las necesidades en cuanto a software no pueden se consideradas del mismo modo, agrupando a todas las empresas bajo una mismo paraguas de necesidades funcionales. Es lógico pensar que una pyme con menos de diez trabajadores (micro empresa) demande funcionalidades distintas a los sistemas de información, que una pyme de 300;

no sólo en complejidad sino también en tipo de procesos. Según vimos, cuatro de cada cinco pymes son micro empresas; este extremo inferior de la amplia categorización pyme es la que más difícil (sino imposible) la adopción de SaaS. Resumimos cuales son los principales escollos para las pymes pequeñas:

- **Precio**: Son muchas las pymes que todavía no pueden permitirse los precios de este modelo.

- **Costes ocultos**: Existen cargos adicionales por parte de los proveedores que encarecen el precio de la aplicación, ocasionando la interrupción del servicio.

- **Complejidad**: Aunque una pyme pueda permitirse el coste de la aplicación, esta puede no satisfacer las necesidades del cliente. La mayoría de las aplicaciones SaaS están diseñadas para las grandes empresas y no para pymes que pueden considerarlas muy complicadas. Su uso puede entorpecer el trabajo en vez de ayudar.

- **Personalización de la aplicación**: La adaptación de las aplicaciones SaaS y configuración por metadatos requieren unos conocimientos de informática que probablemente las pymes no tengan. Además, encargar a un tercero esta personalización encarece de nuevo los costes.

- **Cercanía del proveedor**: Las pymes están acostumbradas a trabajar con empresas locales. Se sienten más cómodas con pequeñas empresas conocidas con las cuales han trabajado desde años, prefieren tener enfrente a personas a las que puedan mirar a la cara. Sin embargo, las empresas proveedores de SaaS suelen ser grandes empresas multi-nacionales con call centers telefónicos que no pueden ofrecer este tipo de servicios personalizados.

Las pymes proveedoras por su parte también tienen dificultades para cambiar la fórmula tradicional on-premises y comercializar SaaS. Los inconvenientes detectados son los siguientes:

- **Alcanzar larga estela**: El éxito comercial de SaaS radica en la escalabilidad, en tener un gran alcance comercial y vender barato a mucha gente. Para una

empresa pyme requiere inversiones publicitarias que no pueden acometer. Es por ello por lo que es rara la existencia de pymes proveedores SaaS.

- **Mano de obra**: Relacionada con la anterior, la escalabilidad hace que los programas SaaS requieran más de diez personas en plantilla. Debemos tener en cuenta que no sólo hay que considerar programadores, sino que el personal para el mantenimiento y soporte, incluso de mayor peso.

- **Escasez de soluciones de industria específica:** Las aplicaciones SaaS suelen ser soluciones horizontales que se adaptan a los sectores mediante caras personalizaciones verticales y horas de consultoría. Para cubrir sus necesidades las empresas optan por la contratación de *freelances* que desarrollan aplicaciones a medida sin eficiencia multi-tenant.

- **Limitación geográfica:** Incluso en el extraño caso de que una pyme comercializara una aplicación SaaS específica de industria, su alcance comercial estaría limitado por su ubicación geográfica debido a su escasa capacidad de inversión publicitaria y comercial.

Si MT^2 supone ventajas a todos los actores en cloud computing, a nivel de pyme podemos considerarlas más que ventajas como una oportunidad; una ocasión antes no presente en la que los proveedores pueden adoptar SaaS como fórmula de distribución mientras que los clientes pueden hacer uso de sistemas de información, antes precarios o inexistentes.

MT^2 es especialmente beneficiosa para la pequeña y la mediana empresa ya que soluciona los problemas detectados para pymes en MT tradicional. Explicamos a continuación esta afirmación, detallando tanto el punto de vista del cliente como del vendedor.

Las ventajas que MT^2 supone para las *pyme cliente* podemos resumirlas en:

- **Nueva reducción de precios** que abarata aún más las aplicaciones y posibilita su contratación, antes inasequible.

- Los **costes ocultos** pueden ser negociados con las empresas pequeñas. Los clientes tienen la confianza y fuerza suficiente con sus proveedores locales tradicionales como para evitar este tipo de "malentendidos".

- El target de la soluciones MT^2 hace que las aplicaciones sean de base **menos complejas y sencillas** de usar. Además, una aplicación SaaS MT^2 puede hacerse todo lo compleja que se quiera, articulando estos grados de complejidad con nuevas funcionalidades. Si un cliente quiere una complejidad mayor, bastará con modificar su contrato de suscripción funcional.

- Bajo esta estrategia de base sencilla y escalado funcional para la complejidad, **no es necesaria la personalización de la aplicación**. Con una suscripción funcional sencilla, las micro empresas pueden simplificar sus tareas diarias y mejorar el rendimiento de sus negocios casi sin esfuerzo.

En el caso de los proveedores también se solucionan los inconvenientes detectados:

- El carácter multi-funcional permite el **alcance de la larga estela** como modelo de negocio. El número de clientes potenciales se ve incrementado con el número de funcionalidades; esto permite a los vendedores acceder a empresas cercanas de otros sectores sin necesidad de acometer grandes inversiones.

- Los sistemas MT^2 son más complejos de desarrollar pero más sencillos de mantener. Solo hay una aplicación y esto permite **reducir el número de personas necesarias** para el mantenimiento.

- Las soluciones MT^2 por definición son multi-sectoriales. Una adecuada selección de contrato funcional las convierte en **aplicaciones SaaS de industria específica** que las pymes del sector pueden contratar.

- La **limitación geográfica** no supone un problema. Los proveedores locales tienen un acceso a un mercado multi-target que ya no limita el número de clientes potenciales. Con una sola aplicación software pueden dar servicio a empresas también locales de diferentes sectores.

MT^2 no pretende eliminar la personalización de las aplicaciones, sino hacer más fácil el trabajo diario. Los metadatos están presentes y pueden ser utilizados al igual que en las

aplicaciones SaaS tradicionales. Sin embargo, en las pequeñas pymes estas adaptaciones normalmente no son necesarias o se reducen a la extensión del modelo de datos, lo cual no requiere muchos conocimientos informáticos. Modificaciones más importantes se resuelven mediante la contratación de nuevas funcionalidades que satisfagan los requisitos. Es mas, si la funcionalidad no estuviera presente, siempre puede ser desarrollada, incorporada al portfolio y ofrecida al resto y futuros de tenants del sistema.

Esta propuesta abre un nuevo campo de aplicaciones SaaS multi-funcionales que soluciona los inconvenientes detectados en la **adopción SaaS por parte de las pymes**. Por tanto, MT^2 mejora las aplicaciones MT tradicionales y está **especialmente dirigida a pequeñas y medianas empresas.**

Capítulo VII

Arquitecturas MT^2

Una vez presentado el nuevo enfoque, en este capítulo se profundiza en la propuesta a través de una solución de diseño a nivel de arquitectura. En primer lugar se ha ideado una definición de un marco arquitectónico que sirve como referencia para las AMTs tradicionales. Posteriormente, esta aportación se utiliza como base para ilustrar el modelo general de arquitectura MT^2 (AMT^2), detallando sus niveles en base componentes y relaciones.

1 Introducción

Dado que la propuesta que se presenta es una extensión para arquitecturas MT, es relevante explicar las modificaciones que se requieren sobre la base del modelo AMT tradicional. Sin embargo, no se encuentran muchas publicaciones que expliquen con detalle y de forma explícita las AMTs. Por ello, se ha ideado y definido un modelo de referencia previo.

En el marco teórico se analiza la propuesta de [23], en la que se describe la arquitectura para aplicaciones SaaS MT a alto nivel pero no detalla en profundidad sus componentes. Además de existir pocos trabajos sobre las AMTs, los que hay centran su atención en la aplicación que los tenants ejecutan, esto es, en la arquitectura de la funcionalidad contratada. Sin embargo, ¿qué ocurre con la aplicación que deben ejecutar los vendedores SaaS? Si una aplicación MT es altamente escalable y puede dar servicio a miles de empresas, cada una con sus cientos de usuarios finales… ¿cómo administrar todas esas cuentas?, ¿cómo controlar el rendimiento y balanceo? Como podemos sospechar, es obligatorio tener una herramienta que de soporte a la gestión y que permita gobernar el sistema MT controlando de forma eficiente el entorno.

Otro aspecto fundamental en SaaS es la integración eficaz con otras aplicaciones [5]. Al igual que el caso anterior, la literatura igualmente no profundiza a nivel de arquitectura cómo tratar esta situación. Las AMTs deben de considerar esta posibilidad y proveer mecanismos de integración con servicios de terceros.

Por todo ello, se hace necesario definir un marco de referencia para las AMTs tradicionales que comprenda un nivel de gestión para los vendedores, así como la integración con servicios de otras aplicaciones.

En este capítulo se propone un marco de referencia de arquitectura de alto nivel para las AMTs, que satisface ambos aspectos. Además de suponer una contribución de esta tesis, nos sirve como base para ilustrar e introducir las AMT^2s, describiendo un modelo de arquitectura para nuestra extensión. Se analizarán modificaciones y nuevos componentes a añadir a las AMTs, para convertirlas a multi-target y poder así, entre otras, gestionar diferentes funcionalidades y un contrato multi-servicio para los tenants.

2 Un marco de referencia para las AMTs

Dean Jacobs apunta la necesidad de tener un *framework administrativo* en los sistemas MT que gestione los metadatos de los clientes y controle el balanceo del sistema [15]. Los autores detectan la necesidad de un nivel de gestión, pero no lo detallan y se centran en la capa de datos sin profundizan en otros niveles de arquitectura. En el marco teórico, también hemos expuesto un modelo de AMT tradicional [23], que ilustra componentes y relaciones. En este mismo trabajo, el autor da importancia a la existencia de servicios de soporte a los proveedores en lo que llama "*Shared Services*"; dentro de ellos tiene en cuenta aspectos como monitorización o facturación, pero sin embargo, no los encuadra definiendo un nuevo nivel administrativo y superior en la arquitectura, especificando componentes, funciones e interrelaciones.

Nuestra propuesta aboga por establecer y detallar un marco de referencia que considere las carencias detectadas en las propuestas existentes. Por un lado, la existencia de un ***nivel administrativo*** hasta ahora desatendido en el estudio de las AMTs; un nivel superior de gestión que permita administrar el entorno MT como sistema. Por otro lado, la existencia de componentes en la arquitectura que faciliten la ***integración*** con otros sistemas facilitando a los tenants el acceso a servicios de otros proveedores y/o a realizar tareas de migración.

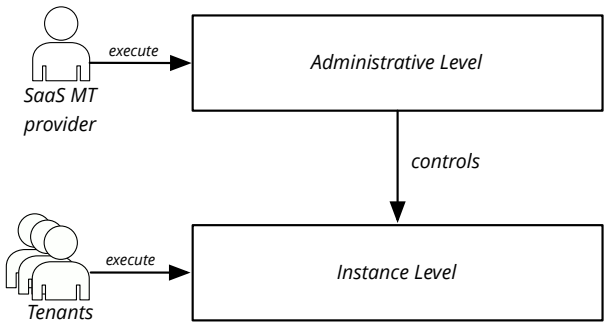


Figura 75. Doble nivel de las MTAs: administrativo e instancia

Por tanto, y según hemos visto anteriormente, en el estudio de las arquitecturas multi-tenant debemos considerar dos niveles generales (Figura 75):

- *Nivel administrativo*: Responsable de la gestión y el control del entorno multi-tenant como sistema. Los datos de este nivel son almacenados en una base de

datos administrativa que almacena entre otras cosas las características de suscripción de cada tenant.

- *Nivel de instancia*: Se trata de las aplicaciones que los tenants comparten y ejecutan. Estas aplicaciones despliegan una determinada funcionalidad (CRM, ERP, SCM) común a todos los tenants. Los datos de los tenants son almacenados en una base de datos multi-tenant (si es enfoque compartido). La aplicación puede ser personalizada individualmente para cada tenant mediante el uso de metadatos a varios niveles (datos, interfaz y lógica). El modelo de datos se puede ampliar mediante el uso de técnicas de mapeo de esquemas en los metadatos de extensión.

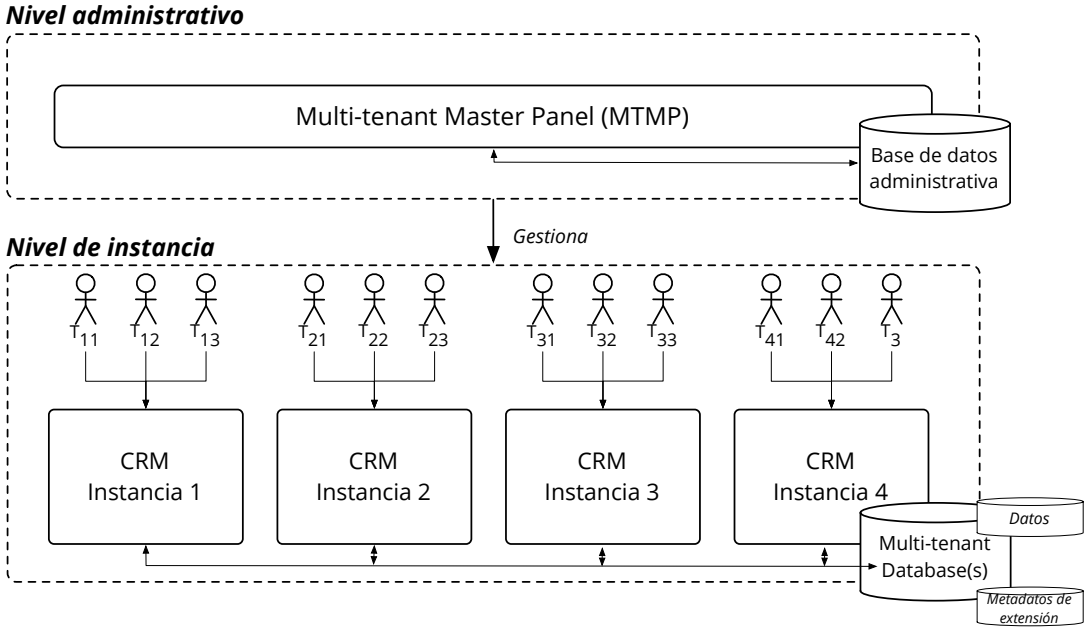


Figura 76. Modelo general de arquitectura SaaS Multi-tenant

La Figura 76 [16] muestra estos dos niveles para un sistema multi-tenant tradicional. El *Multi-tenant Master Panel* (MTMP) representa el nivel administrativo que gestiona y controla el nivel de instancia. El ejemplo muestra en concreto un entorno multi-tenant con una farm de cuatro instancias que despliegan la funcionalidad CRM. El servicio es compartido y ejecutado por doce tenants distribuidos de forma equitativa en la granja. Pasamos a detallar en profundidad cada uno de los niveles.

2.1 Nivel administrativo

El Multi-tenant Master Panel (MTMP) [14] representa el nivel administrativo responsable de la gestión y buen hacer del entorno MT; sus componentes son independientes de las funcionalidad desplegada en el nivel de instancia (ERP, CRM, SCM, etc.). El nivel administrativo puede considerarse como una aplicación software destinada a controlar y gestionar tantos tenants como sea posible, manteniendo su vez niveles adecuados de rendimiento, personalización así como la transparencia en el uso compartido de la aplicación. Con este fin, el MTMP debe atender a una serie de obligaciones con un único objetivo principal: proporcionar buen servicio a los clientes.

Los componentes del MTMP almacenan su información en la *Base de Datos Administrativa* (BDA) y no requieren acceso a la información de los tenants suscriptores, salvo en el caso de copias de seguridad o migraciones.

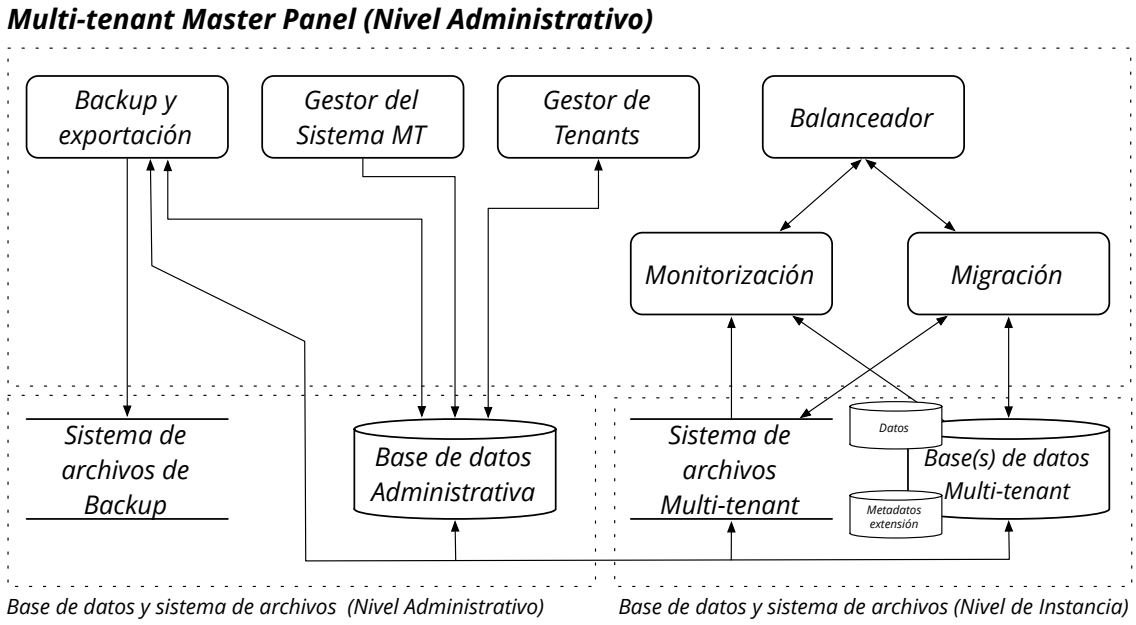


Figura 77. Arquitecturas MT al detalle: Nivel Administrativo

La Figura 77 ilustra los componentes del nivel administrativo de las AMTs; sus responsabilidades son descritas en detalle a continuación

- **Gestor de tenants**. Componente responsable de la creación rápida de nuevas cuentas y de la gestión de las suscripciones de cliente (incluyendo SLAs).

- **Backup y exportación.** Este tipo de operaciones son más importantes si cabe que en los entornos compartidos, ya que un fallo del sistema puede implicar a muchas tenancies. Las tareas deben de ser realizadas para ambos niveles de arquitectura, esto es, no sólo debemos hacer copia de seguridad de los tenants, sino de la información presente en la BDA.
- **Gestor de balanceo**. El control de balanceo sólo es necesario en los entornos de multi-tenancy farm con más de una instancia de aplicación. Con el objetivo de situar a los tenants en el servidor óptimo y lograr un equilibrio de carga, el componente debe de realizar labores de monitorización a nivel de instancia. De este modo, puede examinar la actividad de los tenants, medirla y calcular la carga. Finalmente, el gestor de balanceo es responsable de la migración de las cuentas de cliente a lo largo del farm. El MTMP debe de ser capaz de hacer un *scale-out* de cuentas a (nuevos) servidores [15].
- **Gestor del sistema MT**. Este componente se encarga de las siguientes funciones:
    - Mantenimiento y actualización de *tablas administrativas* comunes que serán compartidas entre todos los tenants (e.g. países, días de la semana, meses, lenguajes, etc.).
    - Definición de los *parámetros de configuración* del entorno MT. Podemos dividir estos parámetros en dos dependiendo de la capacidad de los tenants para modificarlos. La distinción de dónde encuadrar un parámetro, depende del vendedor de software y es definido en este nivel.
        - Parámetros administrativos de sólo lectura para los tenants como por ejemplo cuota del sistema de archivos, conexiones simultaneas, claves de encriptación, paths de instalación, etc.
        - Parámetros configurables de instancia que mantienen un valor por defecto proporcionado en el nivel administrativo pero que puede ser sobrescritos por los tenants en el nivel de instancia. Ejemplos típicos son la paginación de resultados, textos por defecto, calidad de las imágenes tras realizar el upload, etc.

Este nivel de abstracción permite a los proveedores SaaS llevar un control del sistema no sólo a nivel técnico para el mantenimiento y gestión, sino que podemos considerarlo como su base de datos de clientes, y usarlo internamente como módulo de su ERP empresarial.

2.2 Nivel de instancia

Una vez que el nivel administrativo registra una cuenta, el nuevo tenant podrá acceder al nivel de instancia y ejecutar la aplicación desplegada. La existencia de otros tenants en el entorno MT debe pasar inadvertida ya que *"el cliente cree tener una instancia del software entera para él"* [107].

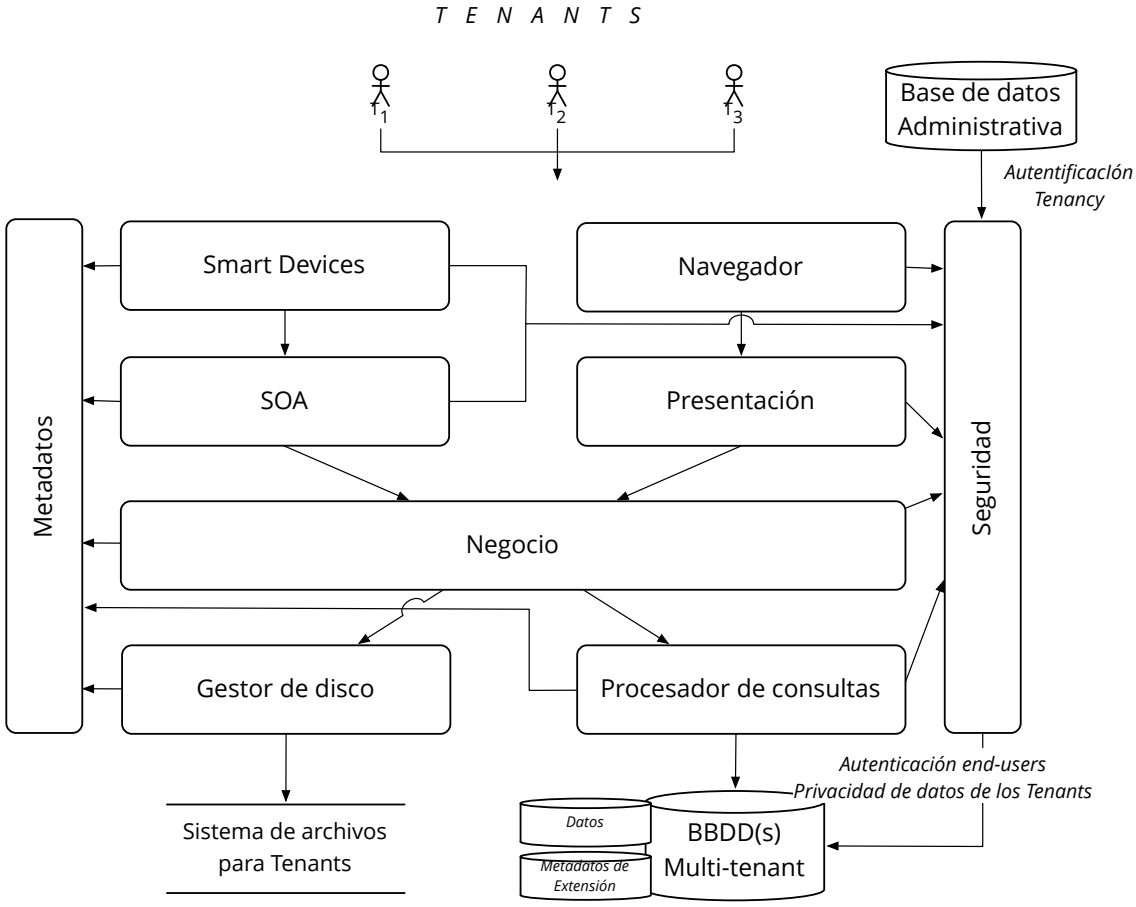


Figura 78. Arquitecturas MT al detalle: Nivel de Instancia.

En la Figura 78 (basado en [23]) podemos ver el nivel de instancia de la arquitectura. Las capas inferiores (más cercanas a infraestructura) llevan a cabo los cambios dictados por el nivel de negocio tanto en la *base de datos MT* como en el *sistema de archivos*.

Dado que se comparte la instancia del software, debemos separar los datos y asegurar la privacidad entre tenants. Por tanto, debe existir una componente *procesador de consultas*, que haga transparente el uso compartido y mapee las consultas lógicas de cada tenant a consultas físicas seguras y eficientes, que extraigan correctamente los datos. El objetivo de la capa de transformación de consultas es modificar las consultas

realizadas por los usuarios finales en su entorno (aparentemente único y no compartido), para adaptarlas al modelo de diseño multi-tenant escogido por el arquitecto software.

Componentes intermedios como la *presentación* o los *servicios SOA* se comunican con el navegador o los dispositivos inteligentes (e.g., smart phones, tablets, etc.) para producir las salidas de los usuarios finales.

Los *metadatos* son los responsables de la personalización del sistema para que los tenants puedan disfrutar de una experiencia única de usuario; esta personalización puede ser realizada en la capa de datos, en la presentación o en la lógica del sistema. A nivel de datos, la personalización en enfoques compartidos se consigue aplicando técnicas de mapeo de esquemas que han sido analizados en el capítulo de Multi-tenancy.

Los servicios de *seguridad* deben de estar presentes en todos los sistemas multi-usuario para garantizar la privacidad y accesos indeseados. Además, esta importancia aumenta considerablemente en entornos multi-tenant ya que son compartidos entre diferentes organizaciones. Por tanto, la complejidad de este componente es mayor.

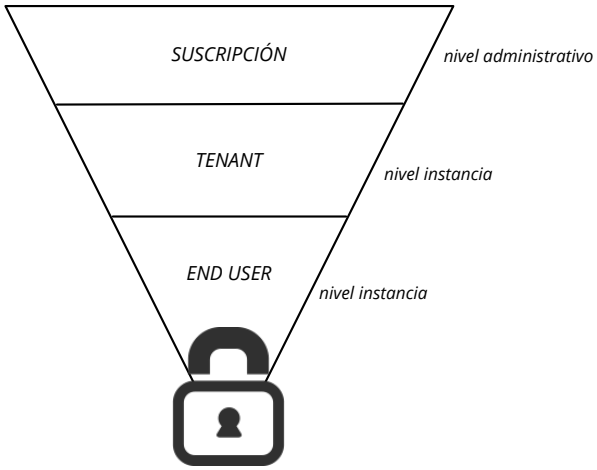


Figura 79. Niveles de seguridad en MT

Tal como muestra la Figura 79, un sistema MT tradicional debe considerar tres niveles de seguridad:

- *Nivel de suscripción*. El nivel administrativo no debe permitir el acceso a usuarios finales pertenecientes a tenants inexistentes o cuentas expiradas.

- *Nivel de tenant*. Tras superar el control de suscripción y ejecutar del nivel de instancia, el sistema debe mantener la privacidad de los datos a nivel tenant. Los usuarios finales de un tenant no pueden acceder a los datos de otro.

- *Nivel de usuario final*. A nivel de instancia, podemos considerar un nivel más refinado de seguridad si la aplicación permite especificar roles o permisos sobre los usuarios de un mismo tenant (al igual que en las aplicaciones multi-instancia).

Como complemento, en el capítulo de Multi-tenancy se han analizado diversas técnicas que permiten asegurar la privacidad en este tipo de sistemas compartidos.

2.3 Integración con otros sistemas

Las AMTs deben de considerar un mecanismo que permita integrar los servicios de otros proveedores. Esto puede ocurrir no sólo porque la aplicación no desarrolle una determinada funcionalidad, sino también por la naturaleza heterogénea de las organizaciones (como las *empresas networking* [195]); además, puede darse el caso de que los tenants deseen migrar su datos a otras aplicaciones. Los sistemas MT deben permitir la importación/exportación de información de/hacia otras aplicaciones.

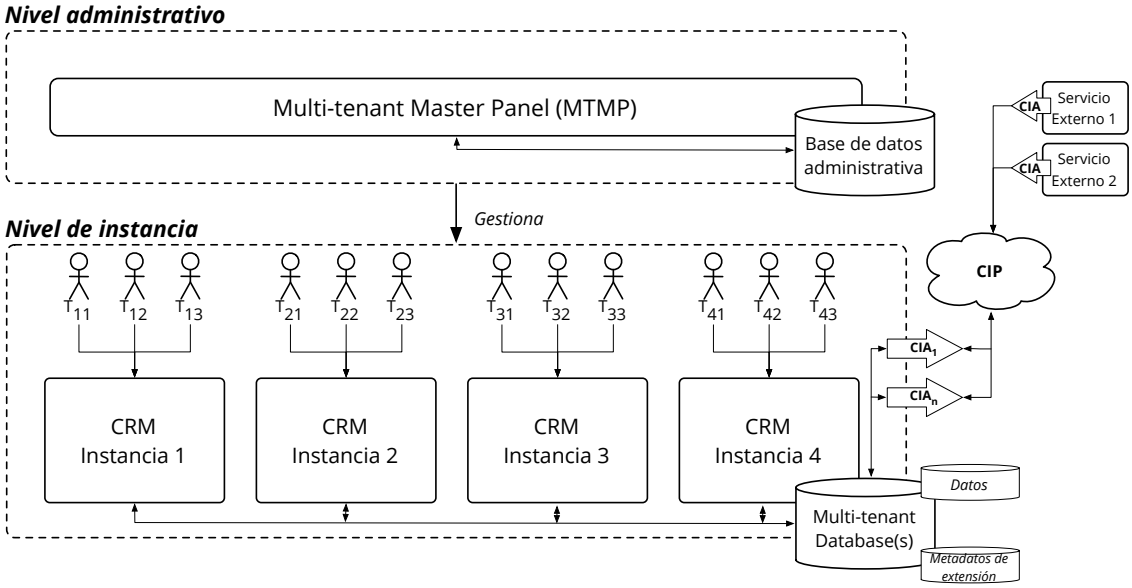


Figura 80. Modelo general de arquitectura SaaS MT con integración de servicios terceros.

Teniendo en cuesta esta preocupación, podemos plantear una modificación a la arquitectura general para permitir esta integración (Figura 80 [66]). Básicamente, la integración de servicios se consigue gracias a *Agentes de Integración Cloud* (*Cloud Integration Agents*,CIAs) presentes en los distintos sistemas y que se comunican entre ellos a través de una *Plataforma de Integración Cloud* (*Cloud Integration Platform*, CIP). La CIP resuelve aspectos clave como contratos de licencia y rutas de acceso al servicio o al negocio [196]. Existen recientes estudios que tratan con mayor profundidad la convergencia de servicios e integración cloud [195]–[197].

3 Arquitecturas MT^2

En MT^2, la arquitectura tradicional es extendida con nuevos componentes y modificaciones sobre elementos existentes, tanto a nivel administrativo como de instancia. Estas novedades tienen por objetivo asegurar el perfecto funcionamiento del sistema en un entorno de suscripción multi-funcional.

Las arquitecturas MT^2 implican no sólo la habilidad de compartir la instancia entre tenants (como ocurría en MT tradicional) sino que también la convivencia de varias funcionalidades. Se hace necesaria la gestión de nuevos metadatos que permitan el despliegue selectivo de las mismas según el contrato multi-servicio (suscripción funcional) de cada tenant.

Siguiendo nuestra propuesta de marco de referencia, al igual que ocurre con los sistemas MT tradicionales, en el estudio de las AMT^2s debemos considerar dos niveles generales:

- Un **nivel administrativo** responsable de la gestión y el control del entorno MT^2 como sistema. Su base de datos administrativa almacena, además de la información presente en su homólogo tradicional, la relacionada con el entorno multi-target (e.g portfolio funcional de servicios o suscripción funcional de los clientes).
- Un **nivel de instancia** que representa de las aplicaciones que los tenants comparten y ejecutan. Al contrario que en los sistemas tradicionales, la funcionalidad desplegada por los tenants no es la misma, sino que depende de los servicios del portfolio incluidos en su suscripción funcional, la cual es definida en el nivel administrativo. Además de a nivel funcional, la aplicación puede ser personalizada individualmente para cada tenant mediante el uso de metadatos a varios niveles (datos, interfaz y lógica). Los datos privados de los tenants son almacenados en una *base de datos* MT^2 y su modelo de datos se puede ampliar mediante técnicas de mapeo. El número de instancias de la aplicación puede ser más de uno, dando como resultado un MT^2 *farm*. A diferencia de los farm MT, las instancias no tienen por qué ofrecer el mismo portfolio ya que depende de las funcionalidades desplegadas por los tenants alojados en cada instancia.

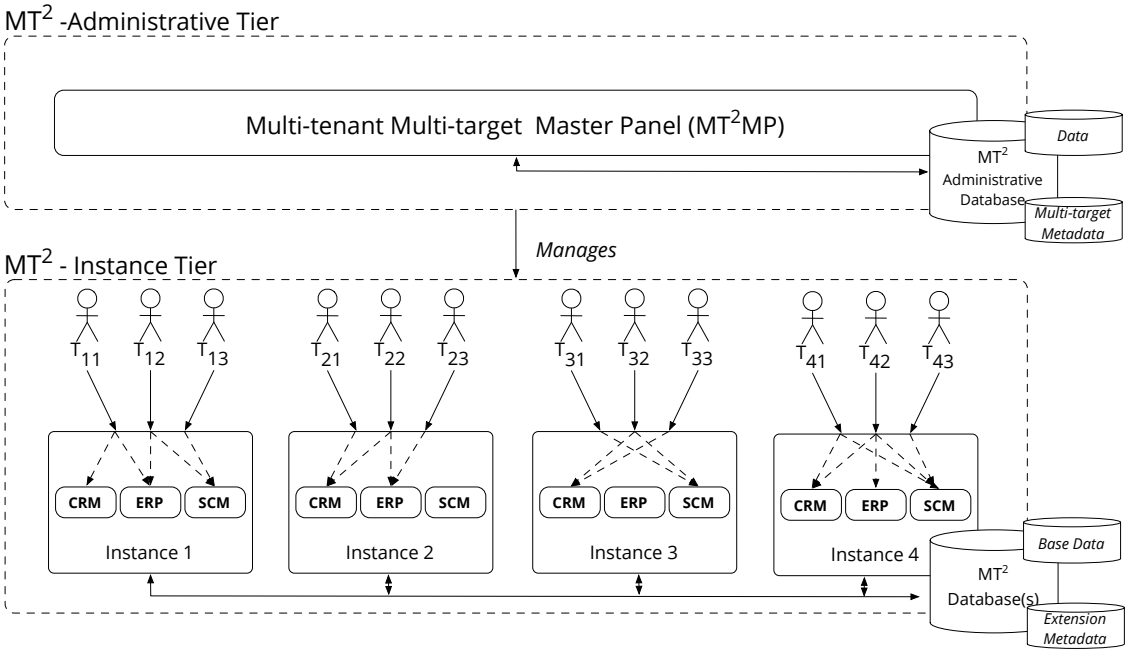


Figura 81. Modelo general de arquitectura MT²

La Figura 81 ilustra el modelo general de arquitectura MT². El escenario representa un farm MT² con 4 instancias y 3 tenants por instancia. Todas las instancias tienen el mismo portfolio funcional (CRM, ERP y SCM) pero no el mismo despliegue. Cada instancia despliega dinámicamente las funcionalidades incluidas en las suscripciones de cliente. Puede darse el caso de que alguna funcionalidad no sea desplegada en alguna instancia por no estar contratada (SCM en la Instancia 2).

El nivel administrativo es representado por el (ahora) *Multi-target Master Panel* (MT²MP). Como faceta extra a su predecesor, el nivel administrativo tiene que gestionar una nueva clase de suscripción multi-servicio en la que, además de los términos presentes en un contrato MT típico, se deben incluir qué funcionalidades del portfolio deben desplegarse. Esta información es almacenada en la *base de datos administrativa MT²*, dentro de los *metadatos multi-target* (*metadatos MT²*). Así, el nivel de instancia sabrá que funcionalidades desplegar por cada tenant, atendiendo al contrato funcional de los tenants extraído de los metadatos MT².

Por consiguiente, en los sistemas MT² **tanto el nivel administrativo como el de instancia presentan diferencias** con respecto a los sistemas mono-target. Además, la gestión multi-funcional ambos niveles tienen que satisfacer nuevos requisitos relacionados con esta novel extensión.

Por ejemplo, los servicios de seguridad deberán velar porque cada tenant despliegue única y exclusivamente las funcionalidades definidas en la capa de administración. En las siguientes secciones estudiaremos con más detalle cada uno de estos niveles y sus disimilitudes con respecto al modelo tradicional.

En esta tesis desarrollamos un capítulo en profundidad sobre el *modelo de datos MT²*, así como otro relacionado con aspectos de *Gestión y Soporte MT²*; sin embargo, en las siguientes secciones expondremos ejemplos introduciendo brevemente pinceladas, que nos ayudarán a clarificar mejor los conceptos. A continuación, pasamos a detallar estos nuevos componentes y actualizaciones.

4 Nivel administrativo MT²

Supone el nivel de gestión del sistema MT². Los usuarios finales que se conectan al mismo, no están definidos dentro de ningún tenant cliente, sino que pertenecen a la empresa propietaria o comercializadora del sistema. Su objetivo principal es el control y el gobierno del entorno MT².

El (ahora) *Multi-target Master Panel* (MT²MP) es una actualización de su homólogo tradicional, al que se han modificando componentes existentes y añadido otros nuevos.

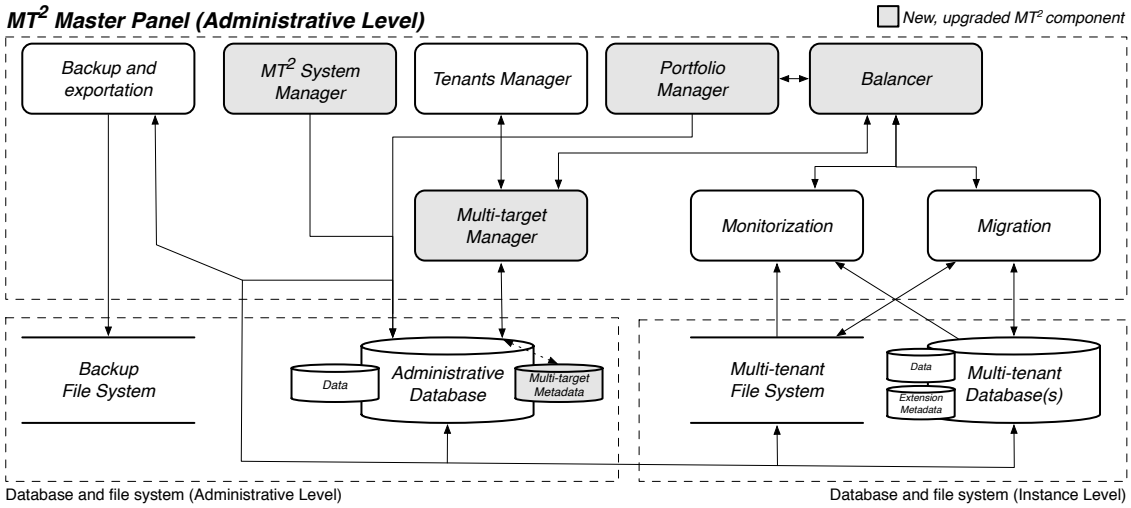


Figura 82. Arquitectura MT² en detalle: Nivel Administrativo

La Figura 82 representa el nivel administrativo MT² con los componentes arquitectónicos afectados oscurecidos; el recuadro inferior derecho de la figura no pertenece a este nivel, sino a componentes de instancia que suministran información

importante a este nivel. Básicamente, los nuevos componentes permiten la gestión de un portfolio de servicios escalable y la administración de las suscripciones multi-funcionales de los tenants. Además, también es necesario modificar otros existentes para adaptarse al nuevo escenario; tal es el caso de gestor de balanceo que ahora debe permitir políticas de migración en función de la carga funcional. De forma resumida, los cometidos del nivel administrativo los podemos dividir en:

- Gestión administrativa de las funcionalidades.
- Gestión de tenants.
- Gestión del contrato de suscripción funcional de los tenants.
- Gestión del balanceo y control de sobrecarga.
- Gestión administrativa del entorno MT2 como sistema.

4.1 Gestor de portfolio

El conjunto de servicios funcionales que un sistema MT^2 ofrece se conoce con el nombre de portfolio funcional. El *Gestor de Portfolio* es el componente administrativo encargado del mantenimiento de las funcionalidades comprendidas en el portfolio.

La *Tabla Administrativa de Funcionalidades* (TAF) registra el portfolio funcional del entorno MT^2. El número y naturaleza de funcionalidades difiere entre sistemas y éstas pueden ser añadidas, eliminadas o actualizadas.

TAF

ID	Nombre
1	ERP
2	CRM
3	CMS
4	DMS
5	SCM
6	SMS

Figura 83. Ejemplo de Tabla Administrativa Funcional (TAF)

La Figura 83 muestra un ejemplo de TAF conteniendo el porfolio ofrecido por un sistema MT^2 concreto. En este caso, el sistema es capaz de desplegar seis

funcionalidades con nivel de granularidad elevado (salvo SMS). Por tanto, los tenants pueden elegir en sus suscripciones el desplegar de forma completa combinaciones de cinco funcionalidades complejas, pero también con la posibilidad de contratar el envío de SMS en cualquier de ellos como valor añadido.

El ejemplo anterior es una muestra sencilla del registro funcional; sin embargo, el mantenimiento del portfolio no se reduce meramente a almacenar el nombre de los servicios desplegados. Existen funcionalidades que por su propia naturaleza precisan tener definidas *parámetros funcionales* que personalicen el contrato de ese servicio concreto. Tal es el caso de la funcionalidad SMS, en la que cada tenant deberá establecer al menos el número de SMS contratados y el precio de cada envío.

Dentro de un farm MT^2, el gestor de portfolio puede estimar la carga computacional de cada funcionalidad y registrarla; de esta forma, *Gestor de balanceo* puede usar esa información administrativa como indicador en sus estimaciones de carga para migraciones.

4.2 Gestor de tenants

El módulo *Gestor de tenants* es el encargado de gestionar las cuentas del cliente y de comunicarse con el gestor de los metadatos MT^2 para dar soporte a las características de suscripción.

La *Tabla Administrativa de Tenants (TAT)* pertenece a la nivel administrativo de la capa de datos y contiene un identificador único de arrendatario (*Tenant_ID*). Además, también debe registrar información de contacto, datos fiscales para la facturación, así como cualquier otra información relevante: servidor de balanceo, idioma, forma de pago, fuente y fecha de captación, etc. (ver Figura 84).

Otro cometido importante de este componente es el *control de expiraciones de suscripción*. El número de clientes en este tipo de sistemas compartidos es muy elevado y por tanto, complejo de gestionar. Para una correcta gestión de suscripciones el Gestor de Tenants debe de comprobar de forma rutinaria y automatizada la expiración de las suscripciones para notificar a los clientes y proceder a una renovación/suspensión del contrato.

Tabla administrativa de Tenants
Tenant_ID
Datos de contacto
Información Fiscal
Otra información Servidor Farm Forma de pago (Mensual, trimestral, Anual) Idioma Divisa Origen de captación

Figura 84. Estructura típica de una TAT

4.3 Gestor multi-target

Además de registrar en los datos de cada tenant en la *TAT*, el nivel administrativo debe almacenar las funcionalidades que contratan. La relación entre los tenants y las funcionalidades está definida en los *metadatos multi-target* (metadatos MT^2)

Además de la propia relación tenant-funcionalidad, el *Gestor multi-target* debe dar soporte a las condiciones particulares de contratación de una determinada funcionalidad. Estas variables de contrato, que determinan las individualidades de suscripción, se representan con la instanciación de los *parámetros funcionales* (anteriormente vistos).

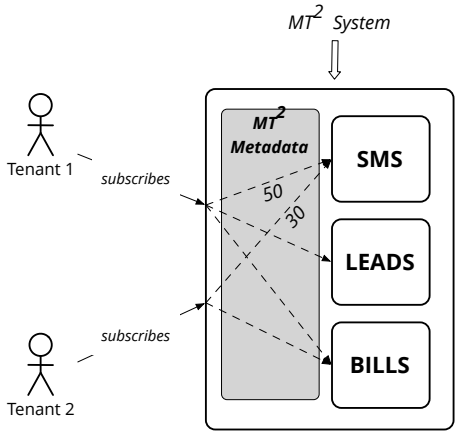


Figura 85. Los metadatos MT^2 definen la suscripción funcional de los tenants.

En la Figura 85 nos encontramos con un sistema de menor granularidad en el que las funcionalidades del portfolio son: SMS, BILLS (facturación) y LEADS (clientes potenciales). Si un tenant quiere suscribirse a la funcionalidad SMS para el envío de mensajes de texto o notificaciones automáticas, al menos deberemos fijar el número de

mensajes contratados. En los metadatos MT^2 (sombreados) vemos como ambos tenants están suscritos a SMS, pero sin embargo el contrato funcional difiere del número de mensajes a enviar (saldo de envío).

Los parámetros funcionales son específicos de cada funcionalidad; no tiene sentido aplicar el número de mensajes contratados en la relación con LEADS o con BILLS del Tenant 1. Así, los metadatos MT^2 para una determinada funcionalidad contienen:

- Relación tenant-funcionalidad.
- Valores de los parámetros funcionales (en caso de que la funcionalidad los tenga definidos).

El gestor multi-target mantiene este nuevo tipo de metadatos MT^2 reflejan no sólo las funcionalidades contratadas, sino también las características específicas de dicha relación. Los metadatos MT^2 reflejan el contrato de *suscripción funcional* de los tenants.

Este componente es una piedra angular para esta extensión y sitúa a MT^2 en el rango de las **meta-aplicaciones**. Las funcionalidades (y por tanto la aplicación) desplegadas en el nivel de instancia dependen del contenido de los metadatos MT^2 y en cuestión de segundos, podemos cambiar por completo el tipo de solución desplegada. El administrador del sistema MT^2 puede hacer que la aplicación de un pase de ser (por ejemplo) un CMS, a un ERP con envío de SMS o un CMS.

Posteriormente veremos cómo los servicios de seguridad a nivel de instancia deben velar por que de la aplicación solamente muestre y ejecute las funcionalidades establecidas en los metadatos MT^2 del cliente.

4.4 Gestor de balanceo

El *Gestor de balanceo* mantiene el equilibrio computacional en el sistema decidiendo en qué servidor alojar nuevos clientes o migrar cuentas existentes. Para ello, es necesario que este componente se ayuda de métricas procedentes de la monitorización del sistema.

El gestor de balanceo debe medir no sólo el grado de uso del sistema, sino el potencial de carga de las funcionalidades contratadas. La gestión de balanceo requiere tener una

Tabla Administrativa de Servidores (TAS) con las características de los mismos (potencia, etc.). Además, la TAT debe también de registrar en qué servidor se encuentra alojada cada cuenta.

La Figura 86 muestra un ejemplo de TAS y su relación con la TAT. En ella podemos ver cómo la mayor parte de suscriptores están alojados en el servidor 1 mientras que los servidores 2 y 3 tan sólo alojan una cuenta cada uno; por su parte, el servidor 4 está ocioso a la espera de nuevas contrataciones o migraciones.

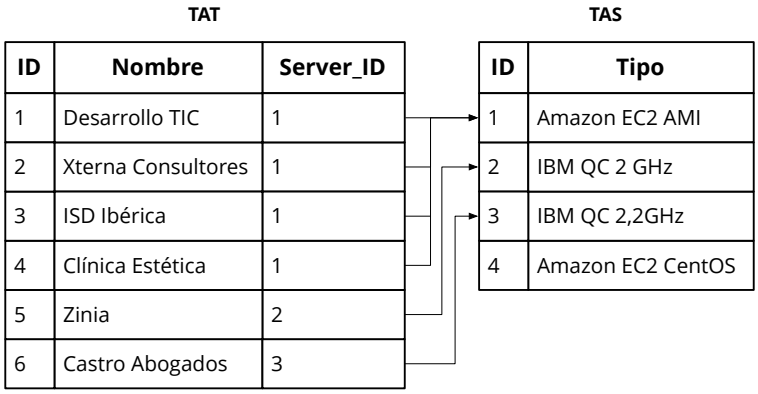


Figura 86. TAS y TAT con nuevo atributo de relación a servidor

Para llevar a cabo su cometido correctamente, el gestor de balanceo ha de contar con las herramientas de monitorización necesarias que proporcionen los datos de uso de cada tenant. El *Gestor de monitorización* es el encargado de proporcionar esta información analizando la actividad de los clientes en el nivel de instancia.

El *Gestor de migración* debe ser capaz de trasladar todos los datos de cliente de un servidor a otro en cualquier momento y hacerlo de manera transparente. Podemos escoger cualquier enfoque de los explicados en el marco teórico:

- Particionamiento horizontal basado en tenants.
- Scale-out de un único tenant.

4.5 Gestor de sistema MT^2

El MT^{2}MP puede ser considerado como una aplicación cuyo cometido es la gestión de un entorno MT^2 y el *Gestor de Sistema MT^2* es el encargado de realizar funciones de

control y configuración. Sus datos son almacenados en la base de datos administrativa y son utilizados por el resto de componentes. Podemos clasificar sus funciones en:

- **Gestión de metadatos administrativos**: configuración del propio sistema MT^2.
    - Parámetros del sistema : Duración de sesiones, conexiones simultáneas, tamaño máximo de archivo, paths de instalación, etc.
    - Idiomas: Definición de idiomas admitidos, contenido del diccionario para las traducciones.
- **Gestión de tablas maestras**: tablas reusables y válidas para todos los tenants
    - Países del mundo.
    - Regiones.
    - Estados.
    - Provincias y Localidades.
    - Días de la semana.
    - Etc.
- **Funciones de exportación y copias de seguridad :** Volcados, sincronización con servidores de respaldo.
- **Obtención de informes y reportes** que ayuden al vendedor a la toma de decisiones (ventas por funcionalidad por ejemplo).

5 Nivel de Instancia MT^2

El igual que en MT, el nivel de instancia se corresponde con las aplicaciones que los tenants ejecutan. Sin embargo, en MT^2 la funcionalidad desplegada no es siempre la misma, sino que depende del *contrato de suscripción funcional* definido en el nivel administrativo. Los usuarios finales dentro este nivel pertenecen a un determinado tenant.

La Figura 87 representa el modelo general de arquitectura MT^2 a nivel de instancia. Las extensiones al modelo tradicional MT aparecen dibujadas con fondo gris. Además, también es necesario modificar el comportamiento de componentes existentes debido al carácter multi-funcional del sistema.

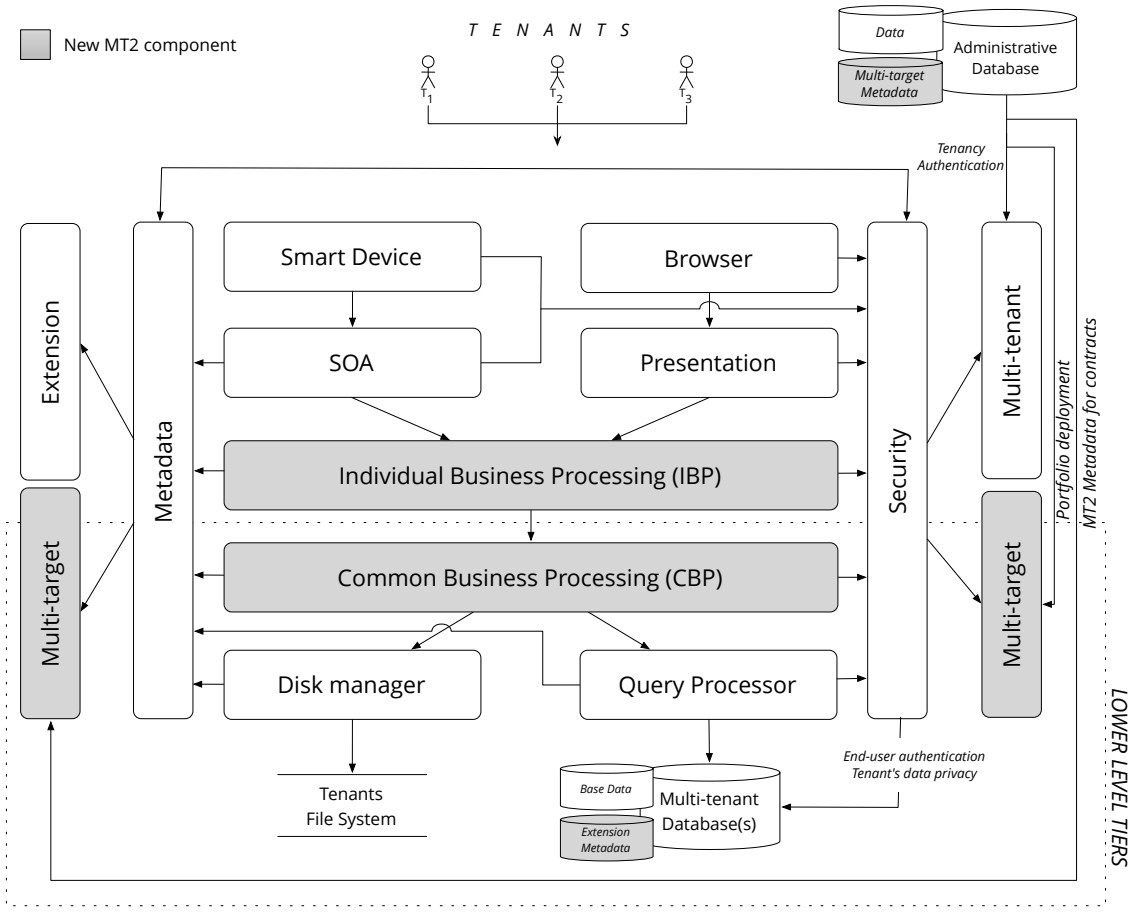


Figura 87. Arquitectura MT² al detalle: Nivel de instancia

El cambio más importante (y que supone la base de nuestra investigación) consiste en la **división del nivel de negocio tradicional en dos** para habilitar la reutilización de componentes entre funcionalidades. Por un lado tenemos un componente de negocio común (CBP, *Common Business Processing*) conteniendo aquellos elementos horizontales y que se pueden compartir entre funcionalidades, y por otro un componente de negocio individual (IBP, *Common Business Processing*) formado sólo por aquellos elementos que son utilizados verticalmente en dicha funcionalidad.

Los componentes de la arquitectura MT² a nivel de instancia los podemos clasificar en cinco niveles:

- **Almacenamiento físico**. Bases de datos de clientes y sistema de archivos.
- **Middleware de información**. Abstracción de acceso al almacenamiento físico.
- **Negocio.** Componentes de procesamiento de negocio.
- **Seguridad.** Encargada de velar el sistema y detener posibles accesos indeseados.

- **Acceso.** Formado por aquellos componentes interfaz para el acceso a la aplicación.

En las siguientes secciones pasamos a ver con detenimiento cada uno de ellos.

5.1 Almacenamiento físico

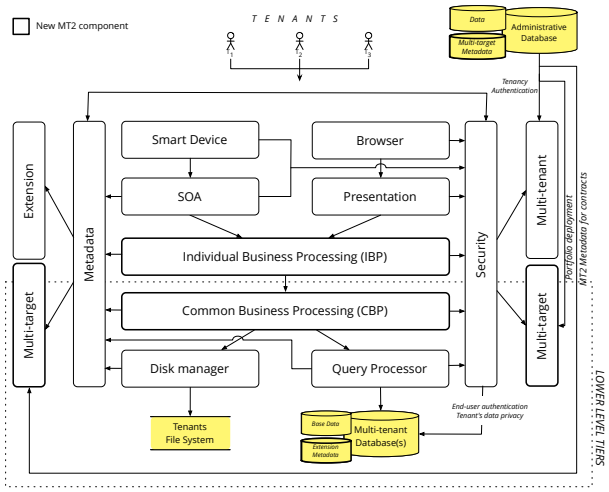


Figura 88. Arquitectura de Instancia MT²: Componentes del Nivel Físico

En el nivel inferior de la arquitectura MT² nos encontramos con las bases de datos y el sistema de ficheros. La base de datos almacena los datos de los tenants y puede tener un enfoque aislado o compartido. En un EIS es deseable que los usuarios almacenen información no estructurada, sobre todo si el sistema despliega funcionalidades como DMS. El sistema de ficheros almacena dicha información documental.

Una característica en las AMTs es la posible personalización del modelo de datos por parte de los suscriptores, añadiendo los campos que necesiten. Así, la información presente en las bases de datos de los tenants (BDT) las podemos dividir en:

- **Información de estructura común:** Contiene información que los usuarios finales registran para cada una de las funcionalidades. Las tablas presentan la misma estructura para todos los tenants.
- **Información de extensión**: Extiende la estructura base sumándole los valores de campos personalizados definidos por los tenants. En posteriores capítulos estudiaremos *Pares MT²*, un enfoque para modelar la extensibilidad en MT².

Existen varios enfoques para modelar la estructura común base que varían en el grado de aislamiento de los datos. En concreto hemos analizado: bases de datos separadas, bases de datos compartidas y tablas compartidas. La elección de uno u otro depende de factores, que han sido analizados en el marco teórico.

La extensión/personalización de la capa de datos puede ser implementada mediante diversas técnicas. En los enfoques aislados, la extensión viene garantizada de manera intrínseca ya que los tenants pueden modificar a su antojo la estructura de las tablas. Para modelos compartidos esta extensibilidad es articulada por técnicas de mapeo de esquemas que permiten traducir las tablas físicas compartidas en las tablas lógicas individuales de cada tenant.

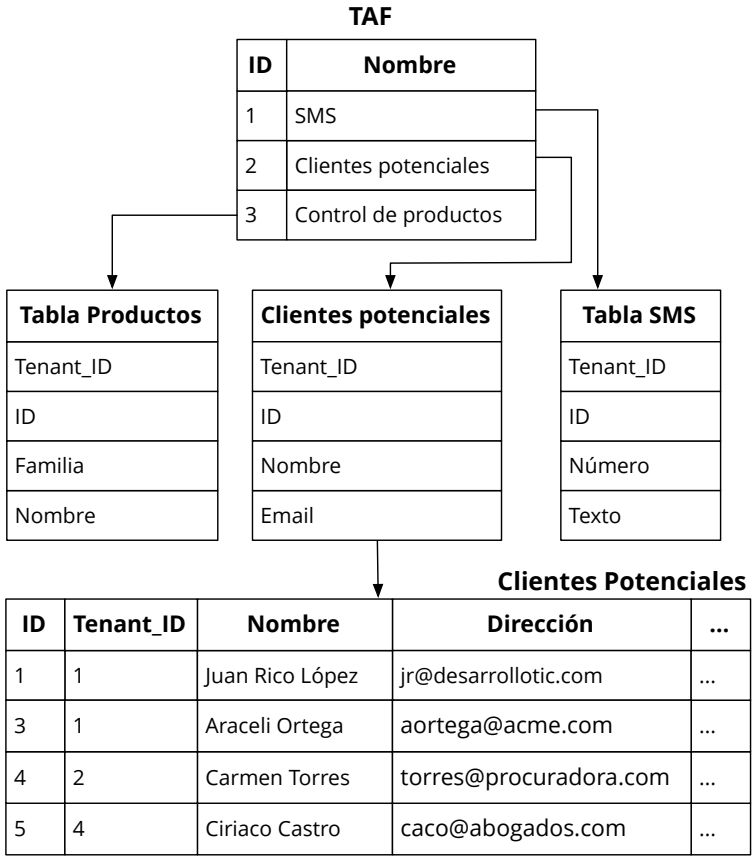


Figura 89. Cada funcionalidad tiene su tabla asociada donde se almacenan los datos de los tenants

El número de *tablas de tenants* (TTs) está directamente relacionado con las funcionalidades definidas en el entorno MT^2. Por cada funcionalidad nueva que se incorpore en el gestor funcional, se creará al menos una nueva tabla. Dicha tabla contendrá los datos de los tenants suscritos al servicio de la funcionalidad. En la Figura

89 vemos un ejemplo de un portfolio de 3 funcionalidades almacenadas en el TAF que necesita al menos tres tablas de información base (sin contar relaciones ni la extensibilidad). En la figura aparece desarrollada con un enfoque de *tablas compartidas* la funcionalidad de *clientes potenciales*.

5.2 Middlewares de acceso a información

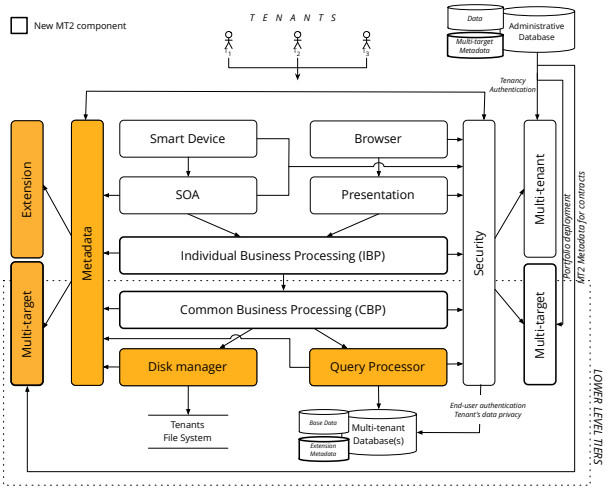


Figura 90. Arquitectura de Instancia MT²: Componentes del Nivel de datos

Los componentes de este nivel son los encargados de abstraer el acceso a la capa de datos y disco. Su objetivo es hacer transparente la estructura de los datos y ficheros de arrendatario. De este modo, simplifican niveles superiores eliminando programaciones complejas de filtros y *joins* en la obtención de los registros y archivos de los tenants.

5.2.1 Gestor de disco

Los usuarios pueden almacenar información no estructurada (archivos) en el sistema y los ficheros almacenados (imágenes, videos, documentos, etc.) están guardados físicamente en disco. Sin embargo, por motivos de seguridad es necesario saber a qué tenant pertenecen los archivos y por tanto, probablemente el enfoque escogido en la implementación precise de tablas en la BDT donde almacenar esta información. El gestor de disco es el encargado de:

- Almacenar y recuperar los archivos del sistema de ficheros.
- Mantener las tablas de la BDT conteniendo los metadatos de los mismos.

5.2.2 Componente de transformación de consultas

Las capas de proceso deben trabajar sobre tablas lógicas y no sobre físicas. El *Componente de transformación de consultas* (CTC) es el encargado de mapear consultas lógicas, haciendo transparente a niveles superiores la verdadera configuración física de la base de datos (Figura 91).

Figura 91. El CTC abstrae a los programadores del entorno compartida.

A nivel de negocio, los programadores pueden trabajar contra este componente, sin tener en cuenta el entorno MT en el que se encuentran. El CTC transformará las consultas que realicen, añadiéndole la privacidad necesaria y modificando las *queries* finales sobre la base de datos física.

La Figura 92-a representa una TT física con un enfoque altamente compartido (*shared tables*) para la funcionalidad Clientes Potenciales. En ella, los datos de diferentes clientes se almacenan físicamente en la misma tabla, manteniendo el identificador de tenant en cada registro. La CTC se encarga de que el entorno compartido de la aplicación sea transparente a los arrendatarios, manteniendo la privacidad de los datos. Así, ante una solicitud de listado de cliente, el CTC transformaría una consulta lógica:

```
select * from clientes_potenciales;
```

En la consulta física:

```
select * from clientes potenciales where Tenant_ID = 1
```

De este modo obtendremos la tabla lógica del tenant 1 (Figura 92-b)

ID	Tenant_ID	Nombre	Mail	Teléfono	...
1	1	Juan Rico López	jr@desarrollotic.com	555-555-555	...
2	3	Fran Pisto Las	frnap@acme.com	555-555-555	...
3	1	Araceli Ortega	aortega@acme.com	555-555-555	...
4	2	Julio La Molda	lamolda@cabezas.com	555-555-555	...
5	4	Ciriaco Castro	caco@abogados.com	555-555-555	...

a)

ID	Nombre	Mail	Teléfono	...
1	Juan Rico López	jr@desarrollotic.com	555-555-555	...
3	Araceli Ortega	aortega@acme.com	555-555-555	...

b)

Figura 92. Tablas física (a) y lógica (b) lógica de la funcionalidad Clientes Potenciales

El CTC debe tener en cuenta la personalización la capa de datos, es decir, los campos definidos por cada tenant. Estos *metadatos de extensión* estarán basados en un determinado enfoque de mapeo de esquemas y la recuperación lógica de las tablas dependerá del mismo. Por tanto, la transformación es un proceso más complejo que un simple filtrado por identificador de tenant. El CTC debe de realizar operaciones join sobre las tablas de extensión para que cada arrendatario obtenga sus tablas lógicas completas. En este trabajo se ha propuesto un modelo de extensión para sistemas MT^2 que se estudiará más adelante.

5.2.3 Gestor de metadatos

Componente transversal encargado de la gestión y mantenimiento de los metadatos a nivel de instancia así como el acceso a los metadatos MT^2. Sus funciones las podemos enumerar en:

- Mantenimiento de metadatos de extensión e implementación de la personalización de la capa de datos.
- Comunicación con el CTC para obtención de las tablas lógicas de usuario
- Acceso a los metadatos MT^2 para la importación dinámica de los procesos de negocio individuales.

- Sincronización con seguridad para evitar accesos funcionales no permitidos.

5.3 Nivel de negocio

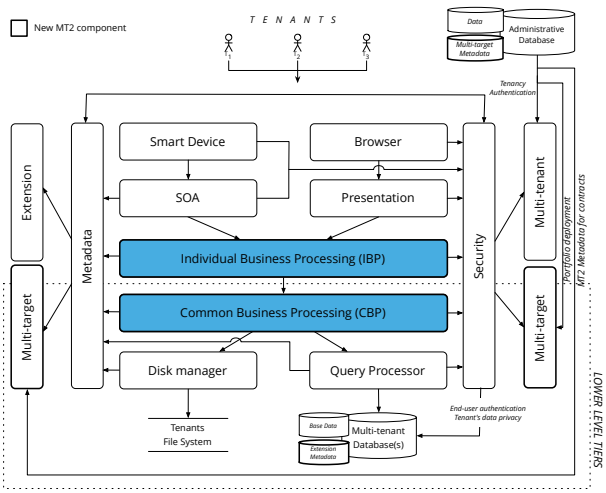


Figura 93. Arquitectura de Instancia MT²: Componentes del nivel de Negocio

En nivel de negocio y sus componentes son la base de la reutilización sobre la que se asienta nuestra investigación. Los integrantes de esta capa se encargan de desarrollar la lógica del proceso de negocio en el sistema. A diferencia de los sistemas MT tradicionales, los componentes los podemos dividir en dos tipos:

- Proceso de negocio común (**CBP**, *Common Business Processing*).
- Proceso de negocio individual (**IBP**, *Individual Business Processing*).

5.3.1 Componente CBP

En un nivel inferior, el CBP desarrolla los procesos de negocio comunes a todos los sectores. Constituye la principal base para el nacimiento de la extensión multi-target. En ella se desarrollan todas aquellas funcionalidades que puedan ser reutilizadas en la capa superior de procesos de negocio individuales.

Gracias a esta reusabilidad, conseguimos dar soporte a la **agilidad en el desarrollo** al no tener que ser programadas de nuevo en las líneas de negocio individuales.

Los elementos CBP son importados de forma estática por todos los tenants ya que desarrollan funciones comunes a todos las funcionalidades. **Ejemplos** de elementos CBP son:

- Abstracciones de **acceso a la capa de datos** sobre el CTC.
- Funciones y librerías de diferente naturaleza:
    - Matrices o cadenas: búsquedas, ordenación, etc.
    - Matemáticas: porcentajes, análisis, etc.
    - Formateo y exportación de datos : RSS, Excel, CSV, iCal, PDF.
    - Comunicación: Envío de mail, notificaciones Ajax Reverse.
- Hojas de estilo.
- Elementos gráficos o multimedia.
- …

Como vemos, los elementos CBP pueden ser de diferente naturaleza, pero todos ellos tienen una premisa en común, la reutilización en capas superiores de la AMT^2.

5.3.2 Componente IBP

El componente de procesos de negocio individuales (IBP), desarrolla a lógica propia de una funcionalidad particular, no común a otras. Al igual que los elementos CBP, un elemento IBP puede ser una hoja de estilos, elementos gráficos y multimedia, funciones, clases, etc. La diferencia con respecto a los elementos CBP radica en que solamente serán importados por aquellos tenants que estén suscritos al servicio concreto que desarrolle esta funcionalidad.

Un ejemplo lo tenemos en la programación de una clase nueva en un lenguaje orientado a objetos. Lo normal es que la clase IBP *extienda* una CBP existente adaptándola a las necesidades concretas de la funcionalidad; esto reduce la carga de trabajo, los tiempos de desarrollo y por ende, **disminuye el time-to-market.**

La capa de presentación también puede contener elementos IBP, únicos por funcionalidad. Por ejemplo, cada funcionalidad puede tener asociado un icono para representarla y que será usado sólo por tenants que estén suscriptos a la misma. Por el contrario, otros recursos gráficos como un icono de descarga o de eliminación pertenecen a la CBP y podrán ser usados por todas las funcionalidades.

Un elemento IBP será sólo incorporado al sistema cuando sea necesario, evitando así sobrecargas.

Los elementos IBP hacen uso de los CBP y son incorporados al sistema de forma selectiva, optimiza el rendimiento durante la ejecución del sistema. La reutilización de elementos CBP supone una herramienta de apoyo a los programadores, los cuales reducen sus esfuerzos de forma considerable.

Los beneficios del enfoque relacionados con el soporte a la agilidad han sido estudiados en el capítulo anterior y no pasaremos a mayor detalle.

5.3.3 Importación dinámica y estática de elementos de negocio

En una ejecución de la aplicación, todos los tenants comparten los elementos CBP, pero no los IBP. Dos tenants importarán los mismos elementos IBP si y solo si están suscritos a las mismas funcionalidades. Así, en cuanto a importación de los elementos en el nivel de negocio podemos distinguir:

- Una **importación estática** de los elementos CBP común a todos los tenants.
- Una **importación dinámica** por cada suscriptor de aquellos elementos IBP pertenecientes a las funcionalidades de los servicios contratados. Para ello debe existir una estrecha comunicación entre la capa IBP y Gestor de metadatos para que la importación dinámica solamente contenga los componentes IBP de las funcionalidades contratadas.

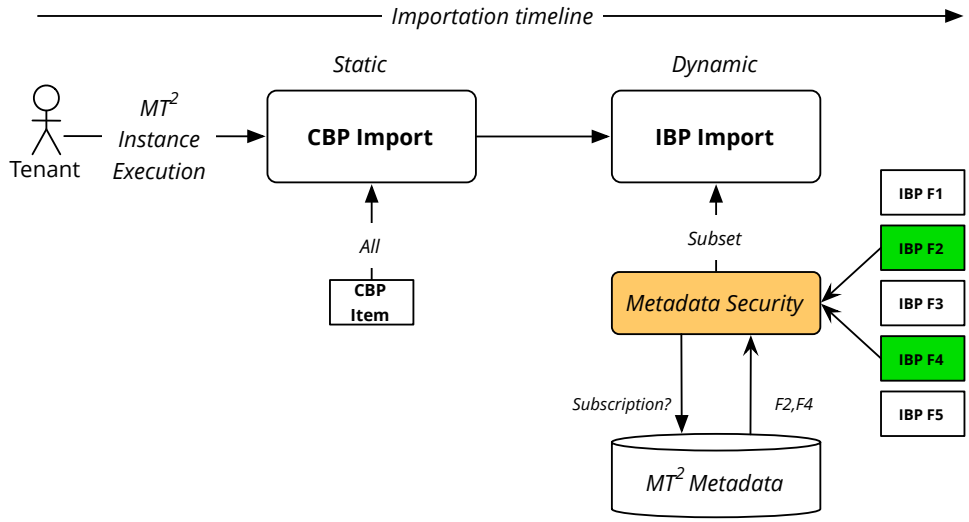


Figura 94. Importación estática y dinámica de elementos del proceso de negocio

El ejemplo de la Figura 94 ilustra la importación dinámica y estática. En la ejecución de instancia todos los elementos CBP son incorporados de forma estática; por el contrario, no todos los componentes IBP serán importados, sino que sólo aquellos pertenecientes a funcionalidades dentro del contrato funcional del tenant. Para saber qué componentes IBP importar, el *Gestor de metadatos* obtiene las funcionalidades contratadas de los Metadatos MT^2 (en este caso la F2 y F4). Por tanto sólo los componentes IBP F1 y IBP F3 serán importados.

5.4 Nivel de seguridad

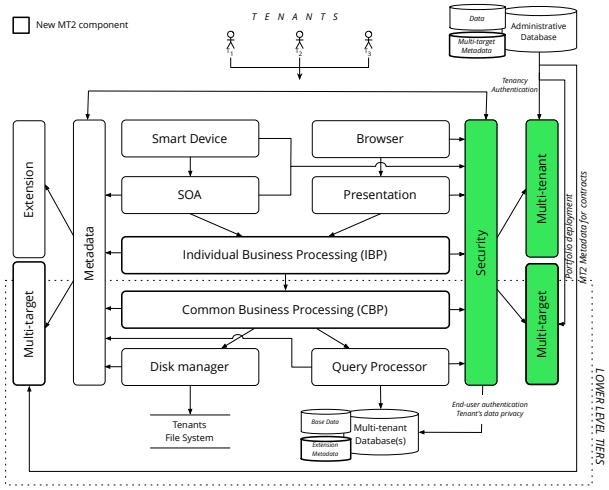


Figura 95. Arquitectura de Instancia MT^2: Componentes del Nivel de Seguridad

La seguridad y la privacidad son junto con el precio el principal factor considerado por las empresas para la migración de las aplicaciones a la nube; por tanto, este componente transversal debe de ser especialmente cuidado y meditado. Las tareas de los componentes de seguridad de la arquitectura podemos agruparlas en los siguientes niveles:

- Nivel de suscripción.
- Nivel multi-target.
- Privacidad de los datos a nivel tenant (multi-tenant).
- Privacidad de los datos a nivel de usuario final.

En la Figura 96 los vemos representados; los dos primeros son ejecutados en el nivel administrativo mientras que los dos últimos se realizan en el nivel de instancia de la aplicación.

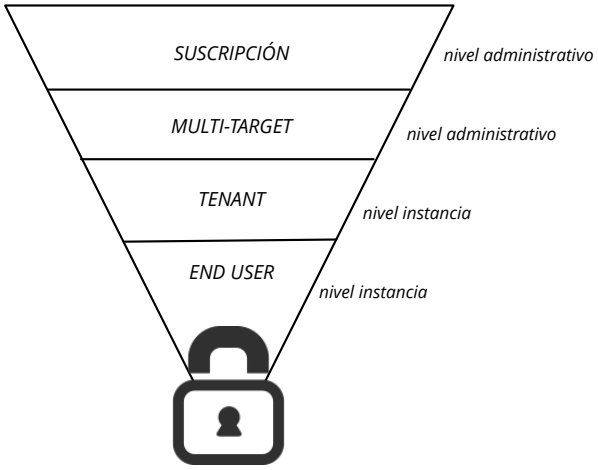


Figura 96. Niveles de seguridad en MT²

La diferencia principal con respecto a los niveles de seguridad MT se encuentra en el nivel 2, ya que la suscripción de cliente implica las funcionalidades a desplegar. Pasamos a detallarlos.

5.4.1 Nivel de suscripción

El nivel administrativo debe impedir accesos al sistema de cuentas de tenants inexistentes o expiradas. Generalmente, en aplicaciones MT las credenciales están divididas en:

- Tenant.
- Nombre de usuario.
- Contraseña.

Como vemos, estos entornos compartidos implican añadir una variable más (identificado del tenant) para identificar al usuario final. Una aplicación MT bien diseñada debe de comprobar en primer término que el tenant identificado pertenece a las cuentas de suscripción en vigencia; tenants inexistentes o pertenecientes a suscripciones expiradas u otros factores (deudores por ejemplo), deberán ser descartados.

5.4.2 Nivel multi-target

La privacidad Multi-target consiste en evitar el acceso a servicios no contratados por el arrendatario, esto es:

- Evitando la aparición de los mismos en la ejecución a nivel de instancia
- Mostrando errores de acceso en la API de acceso a *Third Party Applications*.
- Impidiendo la extensión del modelo de datos de funcionalidades no contratadas.

Para ello, el componente de seguridad debe tener una comunicación constante con el *Gestor de Metadatos* para comprobar que las funcionalidades permitidas al usuario sean las correspondientes a su contrato.

Por ejemplo, en la Figura 97 el tenant 1 solicita un listado de clientes potenciales. Antes de consultar la TT de dicha funcionalidad, la seguridad a nivel multi-target consulta al *Gestor de metadatos* si el arrendatario está suscrito a la funcionalidad para mostrar o no un error de acceso.

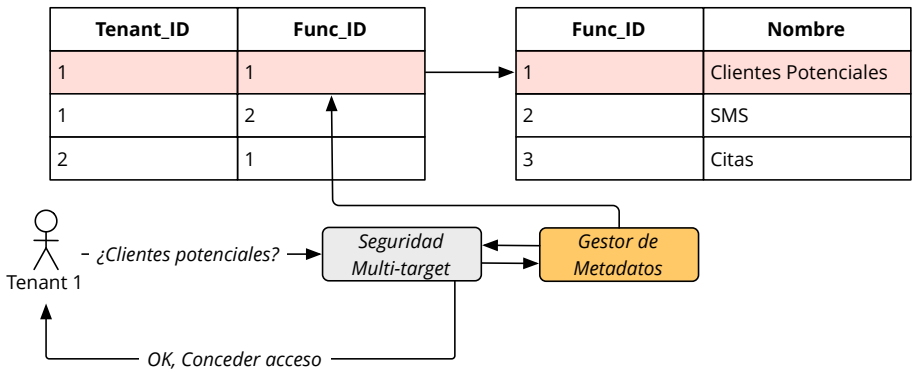


Figura 97. Ejemplo de seguridad Multi-target

5.4.3 Nivel multi-tenant

Se encarga de mantener la privacidad y evitar accesos indeseados entre tenants. Un suscriptor no sólo es incapaz de acceder a los datos de otros, sino que también tiene que desconocer por completo de la existencia de otros suscriptores.

Tal y como hemos estudiado, el CTC se encarga de transformar las consultas de los tenants, de tal forma que accedan únicamente a su conjunto de datos. Aunque pueda parecer que con este componente tenemos solventada la privacidad a este nivel, no es

suficiente. La seguridad debe estar reforzado con comprobaciones de pertenencia de los registros que nos aseguren la propiedad del mismo al arrendatario que los solicita. Recordemos que el acceso a registros puede hacer a través de una aplicación externa con el uso de la API por SOA (por ejemplo). Así en el ejemplo de la Figura 98 un arrendatario con *Tenant_ID* 1 solicita el acceso a un registro de cliente con *Cliente_ID* 5. La seguridad del sistema debe comprobar que dicho registro pertenece a su conjunto lógico de arrendatario. Vemos como efectivamente el registro 5 pertenece al arrendatario 1, por lo que la seguridad permitirá el acceso

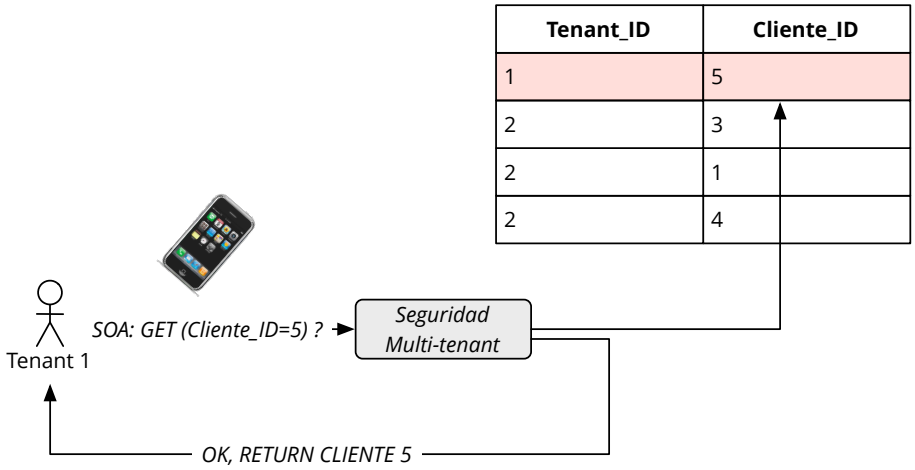


Figura 98. La seguridad MT debe comprobar la pertenencia del registro en cada petición.

5.4.4 Nivel de usuario final

La ejecución de instancia crea un entorno multi-usuario dentro de un mismo tenant. Un tenant puede tener múltiples usuarios finales y estos no deben poder hacer todo lo que deseen dentro del sistema. Sus privilegios de acceso vendrán dados por una seguridad a nivel de usuario que se encargue de:

- *Evitar accesos indeseados* a registros dentro del mismo tenant. Un usuario final sin permisos, no podrá borrar los registros de otro por ejemplo.
- *Omitir funcionalidades* a determinados usuarios. Aún teniendo la funcionalidad contratada, puede que al usuario administrador del tenant le interese ocultar determinadas funcionalidades a otros usuarios.

La omisión de funcionalidades a usuarios finales es algo lógico. Por ejemplo, pensemos en un contrato de servicios que incluya facturación, clientes potenciales, y CMS; a la

directiva de la empresa, le interesa que los usuarios con acceso a facturación sean únicamente aquellos del departamento de contabilidad. Para ello, la seguridad a nivel de usuario final mostrará la funcionalidad sólo a aquellos trabajadores con permisos de acceso a la misma, en su usuario final.

5.5 Nivel de acceso

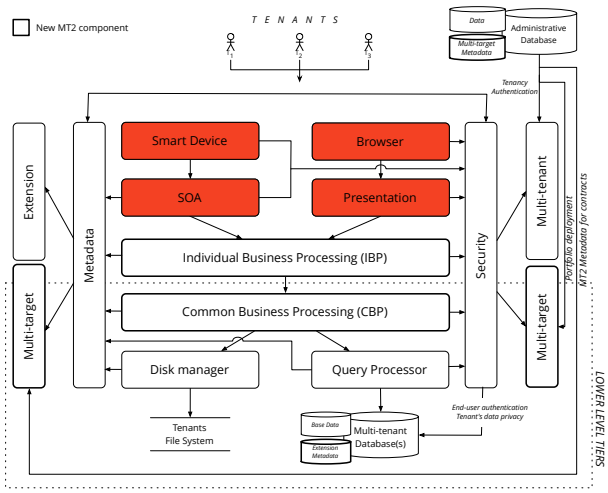


Figura 99. Arquitectura de Instancia MT²: Componentes del Nivel de Acceso

En este nivel, los usuarios finales interaccionan con la aplicación. Podemos distinguir dos formas de acceder al entorno MT²:

- **Navegador**: los usuarios finales se autentican en una página web.
- **Aplicaciones de terceros** (*Third Party Applications*): Aplicaciones desarrolladas por otras empresas.

5.5.1 Acceso por navegador

Supone la principal vía de acceso al sistema. La aplicación MT² dispondrá de una dirección URL para la autenticación dentro del entorno MT². El servidor devolverá en última instancia código HTML que será interpretado por los navegadores de los usuarios finales.

El acceso por navegador yace sobre una capa de *presentación* que define el aspecto de la aplicación. Incluir en una misma página HTML los contenidos, el diseño y la programación complica en exceso su mantenimiento. Las *hojas de estilo* (*Cascade Style Sheets*, CSS) permiten separar los contenidos del aspecto que deben presentar.

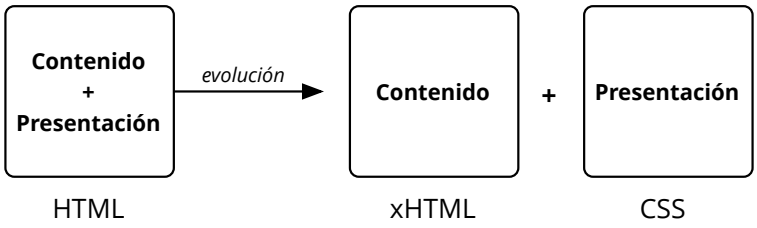


Figura 100. Evolución del HTML para la separación del contenido y el aspecto

La separación del contenido y la presentación (Figura 100) mejora no sólo la facilidad de mantenimiento, sino otros aspectos fundamentales como por ejemplo la accesibilidad. Además, los tenants o incluso los usuarios finales podrán personalizar la aplicación si permitimos que utilicen sus propias hojas de estilo.

5.5.2 Aplicaciones de terceros (*Third Party Applications*)

Los tenants deben poder acceder a los servicios contratados a través de plataformas no desarrolladas por el vendedor original del software MT^2. Hoy en día vivimos el paradigma de la computación ubicua, en la que debemos poder ver nuestras aplicaciones online a través de todo tipo de dispositivos y plataformas. Un enfoque SaaS adecuado de comercialización implica que los tenants que se suscriben a nuestro software con una características y condiciones en el contrato (funcionalidades, usuarios finales, fecha de expiración de servicio, etc.) , pueden acceder a las mismas en cualquier momento y desde cualquier dispositivo.

Un buen sistema debe contar con la capacidad de acceder a funcionalidades en otras plataformas distintas, gracias a la existencia de servicios web y APIs de desarrollo.

6 Integración con otros sistemas

La integración de sistemas es factor de gran preocupación en el dominio del software empresarial. MT^2 da un paso más permitiendo la incorporación de funcionalidades completas en lugar de pequeñas características o integración de datos. Este hecho puede darse bien por la ausencia de la funcionalidad, bien por la preferencia de uso de funcionalidades semejantes presentes en otras aplicaciones.

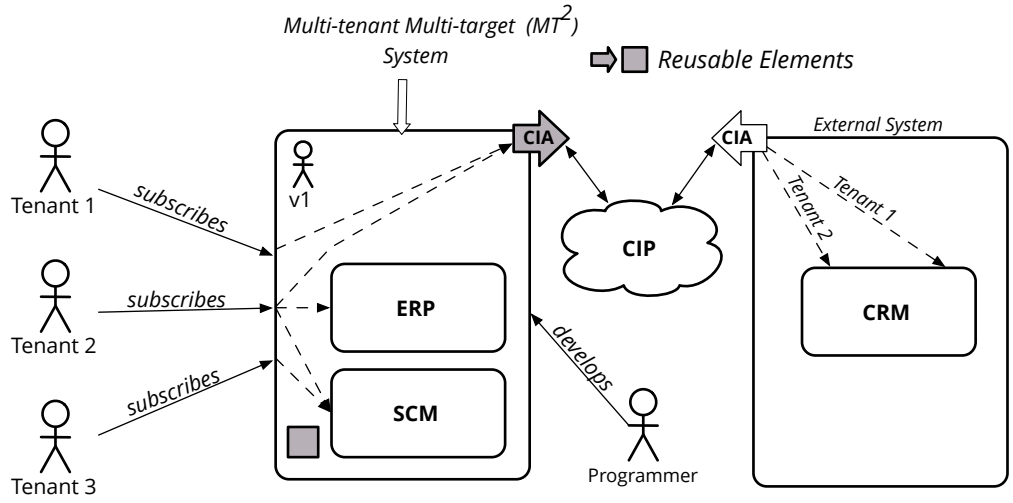


Figura 101. Nuevo escenario MT^2 para la integración de sistemas

La Figura 101 añade un tercer escenario MT^2 en el que la funcionalidad ofrecida es proporcionada por un proveedor externo. Supongamos que el portfolio funcional del sistema MT^2 comprende ERP y SCM; si queremos satisfacer la necesidad de CRM de los tenants 1 y 2 podemos desarrollar la funcionalidad e incorporarla al portfolio, pero el periodo de desarrollo sería elevado. Otra opción que permitiría satisfacer la necesidad funcional sería usar el servicio CRM de otro proveedor externo, integrándolo en la plataforma MT^2. La programación sería más rápida y además permite que un cliente ya suscriptor de ese CRM, pueda usarlo en la plataforma MT^2 sin necesidad de migración. En este caso, los CIAs de los sistemas colaborarían a través de la CIP (*Cloud Integration Platform*) para integrar el servicio como funcionalidad.

A nivel de arquitectura general del sistema, los metadatos MT^2 determinarán si la funcionalidad suscrita está alojada en el sistema MT^2 o en uno externo. Como vemos en la Figura 102, la instancia 4 del sistema MT^2 tiene externalizada la funcionalidad de CRM a un servicio de otro proveedor. A pesar de estar incluida en el portfolio (y

contratada por el T_{42}), el T_{41} prefiere el uso de la aplicación externa. Por cada servicio externalizado, existe un agente de integración que compartirá recursos reutilizables con otros agentes.

Los componentes CIA en sistemas multi-target actúan no sólo a nivel de datos, sino que también a nivel funcional incorporando al portfolio funcionalidades completas de otro proveedor.

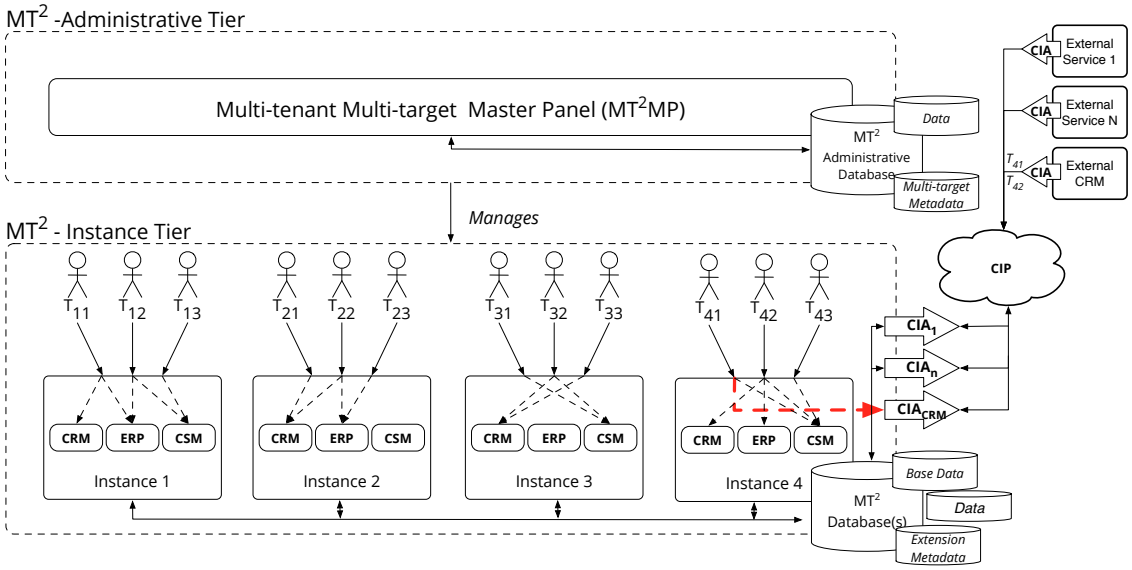


Figura 102. Modelo general de arquitectura MT². Integración con servicios cloud externos

Desde el punto de vista administrativo MT², sería cuestión de añadir ese servicio a la suscripción, teniendo en cuenta la naturaleza externa de la funcionalidad proporcionada. Esta situación implica beneficios para proveedores y clientes; los proveedores siguen haciendo negocio debido a la externalización mientras que los clientes ven satisfechas sus necesidades funcionales (sin necesidad incluso de migración). Además, los componentes CIA de los sistemas MT² forman parte de los elementos reusables por lo que no sería necesario replicarlos, ni programar desde cero.

Capítulo VIII

Bases de datos MT^2

La privacidad y seguridad de los datos es un factor clave para la implantación de SaaS en las empresas; por esta razón, se dedica un capítulo completo al estudio de este nivel en la arquitectura MT^2. En primer lugar, se contribuye definiendo un marco de referencia para el diseño de bases de datos MT tradicionales. Posteriormente, basándose en esta aportación se extiende la idea detallando un modelo relacional para bases de datos MT^2.

1 Introducción

El estudio de la capa de datos MT cobra mayor presencia en la literatura a medida que proliferan las suscripciones de las empresas a sistemas de información en la nube. Además del precio, la privacidad y seguridad en el acceso a la información son los factores de mayor preocupación para las empresas a la hora de trasladar sus aplicaciones tradicionales el modelo cloud [26], [108], [109]; por ello, la capa de datos en entornos compartidos MT es de gran interés para los académicos. En el marco teórico se han estudiado enfoques que permiten implementar multi-tenancy a nivel de bases de datos; existen modelos compartidos o aislados dependiendo del grado de compartición de los datos entre los tenants; además, también hemos analizado distintas técnicas de mapeo de esquemas que permiten a las organizaciones personalizar la capa de datos de la aplicación MT almacenando información extra en metadatos.

Sin embargo, hasta ahora **no hemos podido encontrar trabajos que establezcan un marco de referencia** para el diseño de la capa de datos de las AMTs. Dada la complejidad e importancia de este nivel en la arquitectura, se hacen necesarias guías y herramientas de ayuda, que planteen una división concreta de los modelos necesarios, y faciliten así el trabajo de los arquitectos SaaS en el diseño de la capa de datos. Por ejemplo, cuando diseñamos un sistema MT no es lo mismo el modelo de datos usado para implementar multi-tenancy que el utilizado para habilitar la extensión por metadatos (si existe); pueden estar relacionados o ser dependientes, pero son **decisiones diferentes**.

A diferencia de las aplicaciones multi-instancia tradicionales, las AMTs deben de disponer un nivel administrativo por encima de la arquitectura que permita un rápido acondicionamiento y gestión de los tenants [15]. De este modo, no sólo el precio es el único beneficio, sino que también la reducción del time-to-market y un rápido despliegue. A pesar de la importancia de este nivel, los estudios de la capa de datos existentes **se reducen al nivel de instancia**, esto es, al estudio de enfoques que analizan cómo almacenar la información de los tenants. En ningún momento se habla de la **necesidad de un modelo en este nivel administrativo**, que permita administrar la aplicación y de soporte a un correcto gobierno del entorno MT (control de suscripciones, balanceo, configuración, etc.).

Por ello, antes de presentar un modelo relacional detallado para MT^2, se propone inicialmente un *marco de referencia para el diseño la capa de datos MT tradicional* que nos servirá como base para su posterior extensión al modelo MT^2.

2 Marco de referencia para bases de datos MT

En un entorno multi-tenant, diferentes organizaciones comparten la aplicación pero no sus datos. Por tanto, en el diseño del sistema debemos plantearnos cómo implementar de forma segura el almacenamiento de información, manteniendo la privacidad entre los distintos clientes. Además, es un requisito deseable que los sistemas permitan a los suscriptores extender el modelo de datos inicial para adaptarlo a sus necesidades y sin que estas adaptaciones afecten al resto de los tenants.

Un aspecto muy importante es que los sistemas MT son también aplicaciones en sí que necesitan ser controladas y gestionadas; el nivel administrativo de las AMTs debe llevar a cabo operaciones como gestión de los tenants, monitorización o migración, que necesitan del soporte de una base de datos, con un diseño específico. Este modelo es imprescindible para la gestión de los proveedores SaaS, pero sin embargo es frecuentemente olvidado en la literatura. Según [15], los sistemas multi-tenant deben de ser capaces de ejecutar de forma eficiente *operaciones administrativas masivas* que permitan actualizar las tablas de los suscriptores así como obtener informes relacionados con los datos de los tenants; como vemos, existen autores que lo mencionan, pero no hay trabajos que entren en mayores detalles de su diseño.

Según lo anterior, **en la implementación de sistemas MT tradicionales, debemos considerar tres modelos en el diseño de la base de datos**

- Un modelo *administrativo* para la gestión y soporte al vendedor de la aplicación (Figura 103-a). Este será el modelo correspondiente al nivel administrativo de la arquitectura.
- Un modelo *base* que defina la estructura común para los datos de los tenants (Figura 103-b). El modelo será usado en el nivel de instancia
- Un modelo de *extensión* también en el nivel de instancia que permita a los tenants ampliar el modelo base. Permite configurar la aplicación adaptándola a necesidades individuales (Figura 103-c).

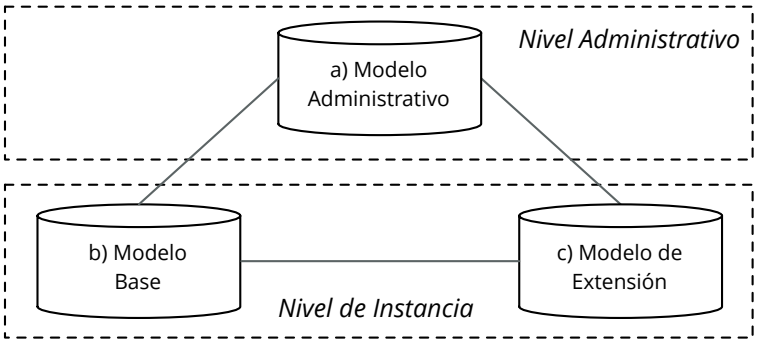


Figura 103. Modelos a considerar en el diseño de una bbdd multi-tenant

2.1 Modelo administrativo

Las aplicaciones SaaS MT se ofrecen a las organizaciones en un régimen de suscripción, según uso o consumo. Ya pagamos con este modelo otros servicios como el agua, el gas, la electricidad o el teléfono y, del mismo modo que las empresas suministradoras utilizan sus sistemas de información para el control y la gestión, los proveedores SaaS necesitan de un nivel administrativo [15], [16], [37] en la aplicación MT con un modelo de datos diseñado para este fin.

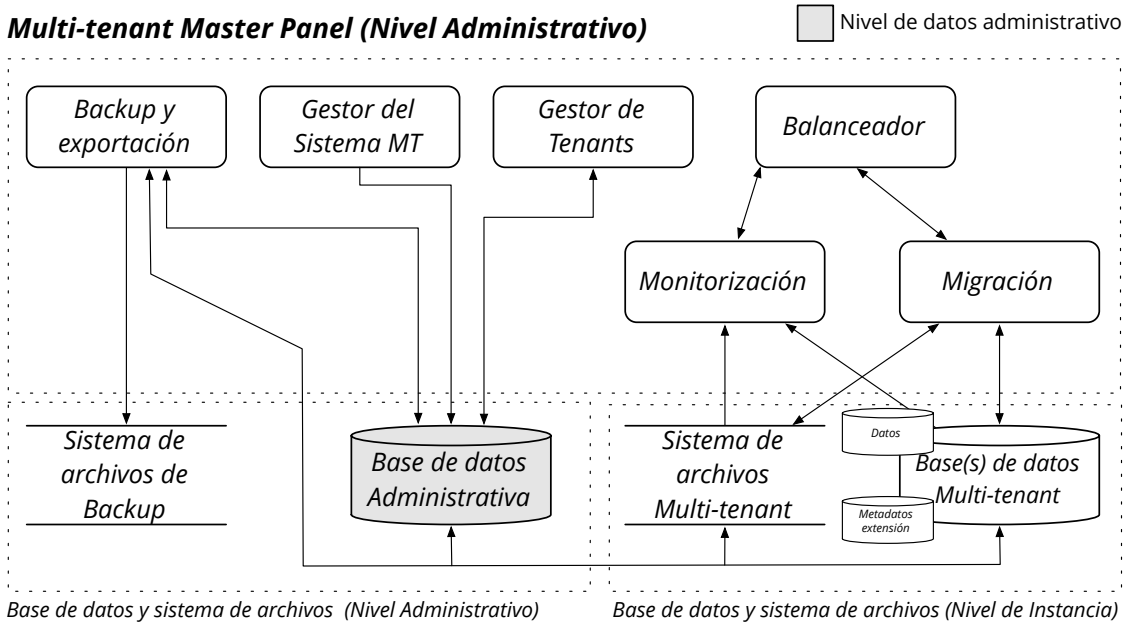


Figura 104. Arquitecturas MT al detalle. Componentes administrativos de la capa de datos

La Figura 104 recuerda el nivel administrativo de la arquitectura, resaltando en gris los componentes de la capa de datos a este nivel. Como podemos ver en la misma figura, existen también componentes relacionados con los datos del nivel de instancia (parte

inferior derecha); el modelo administrativo de este marco de referencia no los incluye, ya que pertenecen al modelo base en el que los tenants almacenan su información.

Este modelo de datos administrativo es usado por los distribuidores SaaS y tiene que dar soporte a aquellas operaciones de gestión necesarias para el buen funcionamiento del entorno MT. Además, debe de considerar otras funcionalidades no relacionadas directamente con multi-tenancy sino con aspectos propios de soporte empresarial, típicos de sistemas de información. Ponemos algunos ejemplos de estas funcionalidades:

- Gestión de tenants y de sus contratos.
- Configuración del sistema.
- Monitorización del rendimiento.
- Control de balanceo (en caso de multi-tenancy farm).
- Gestión empresarial común (facturación, clientes, etc.).

En definitiva, podemos hablar del nivel administrativo MT como el modelo de datos de un *sistema de información empresarial* (SIE) que ayuda a las empresas SaaS a gestionar un producto multi-tenant. En este caso, el producto es un servicio que se consume bajo demanda y son aplicaciones software. Además de la gestión del entorno MT, los proveedores necesitan también herramientas de gestión empresarial típicas encuadradas dentro de tipologías estándar de EIS como ERP, CRM o SCM. Este modelo no implica implementación de multi-tenancy (aunque podría considerarse para el caso de *resellers*).

2.2 Modelo base

El modelo base implementa multi-tenancy y mantiene la privacidad de los datos entre los tenants. La Figura 105 recuerda de nuevo el nivel de instancia, destacando en gris los componentes relacionados con la capa de datos. A nivel de datos, multi-tenancy se implementa transformando las *tablas lógicas individuales* de cada tenant en *tablas físicas comunes* a los suscriptores. Esta conversión es realizada a cabo por el *procesador de consultas*, que abstrae el acceso y simplifica el trabajo a los programadores.

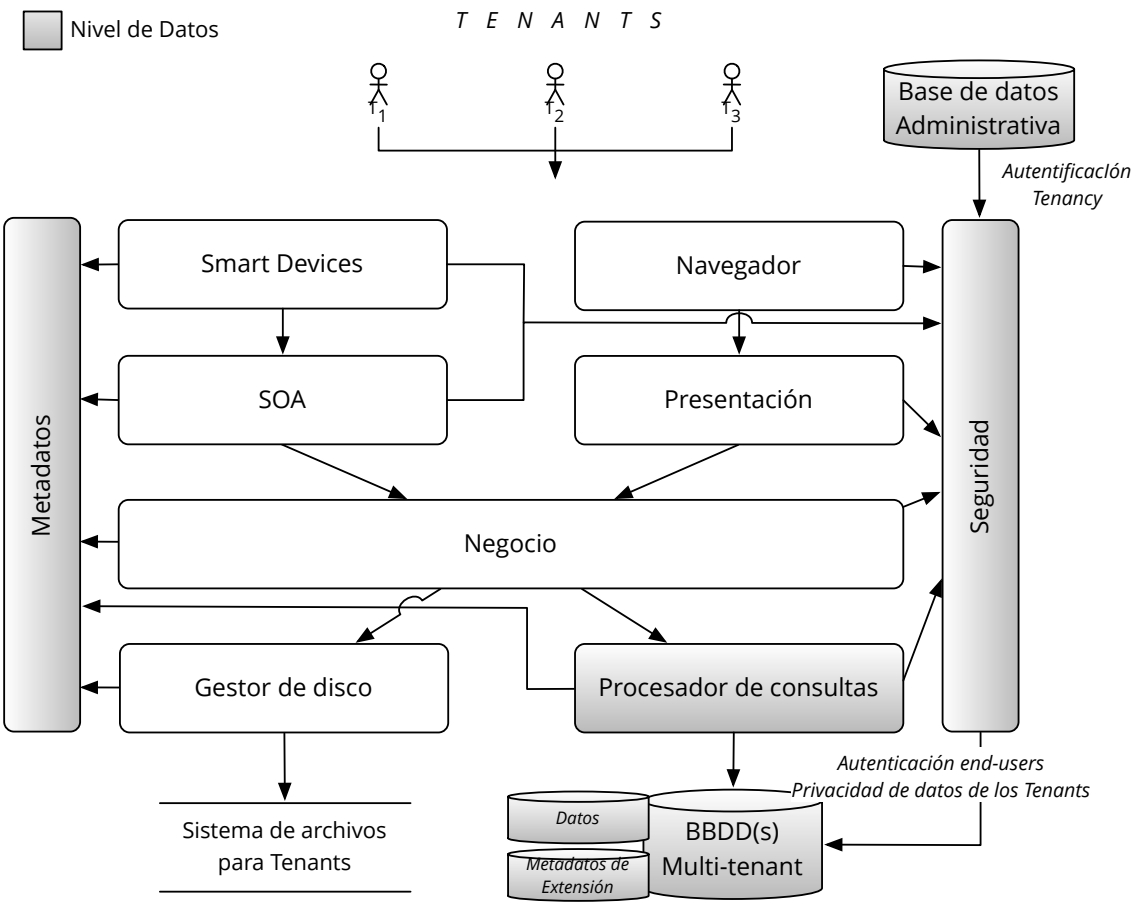


Figura 105. Arquitectura MT al detalle. Componentes de la capa de datos en el nivel de instancia

La literatura considera varios enfoques dependiendo del grado de aislamiento de los datos, esto es, según el nivel de compartición física de la información:

- Alto grado de aislamiento (*Bases de datos separadas* [15], [103]): cada tenant almacena los datos en una base de datos diferente (Figura 106-a)**.**

- Grado medio de aislamiento (*Base de datos compartida* [15], [103]): los tenants comparten la base de datos pero las tablas son distintas. Cada suscriptor tiene un esquema de tablas privado en el que almacenar los datos (Figura 106-b)**.**

- Grado bajo de aislamiento (*Tablas compartidas* [15], [103]): en este caso se comparte tanto la base de datos como las tablas. Una misma tabla del nivel de instancia contendrá información de diferentes tenants. Cada registro deberá llevar un identificar de la organización a la que pertenece (Figura 106-c).

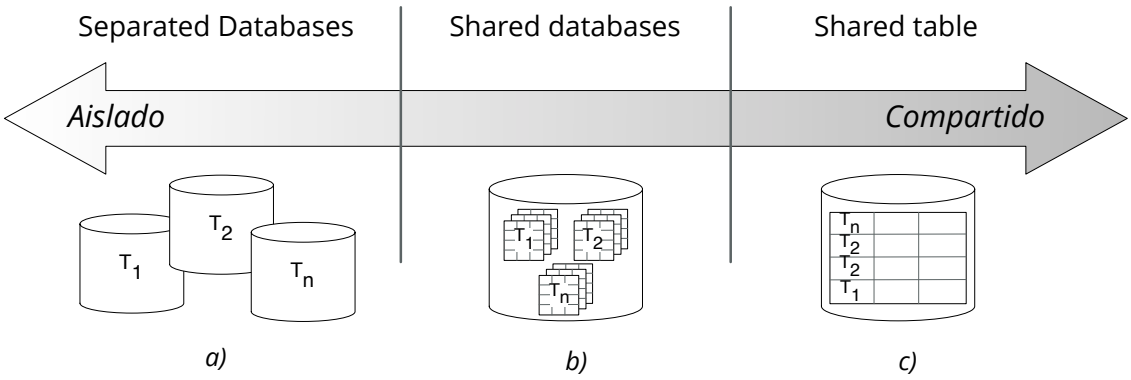


Figura 106. Enfoques de diseño para el modelo base.

El modelo base pertenece al nivel de instancia y determina el diseño común de las tablas en el entorno MT; todos los suscriptores tienen el mismo para almacenar su información. Al dar de alta un cliente en el nivel administrativo, las tablas base son inicializadas según el enfoque escogido.

2.3 Modelo de extensión

Una vez diseñado el modelo base, tenemos un conjunto común de elementos para todos los tenants (tablas, relaciones, *triggers,* etc.). Sin embargo, cada empresa puede tener unas necesidades diferentes, incluso perteneciendo al mismo sector. Nuestra aplicación debe poder permitir a cada tenant tener una personalización de la aplicación SaaS de forma que se adapte a sus necesidades y sin necesidad de duplicar el código.

Supongamos por ejemplo una base de datos MT en la que tenemos una tabla de clientes; una empresa puede necesitar añadir un campo más (número de hijos) y no encontrarse en el diseño base. La aplicación debe permitir al tenant añadir este campo a su modelo sin que aparezca al resto de suscriptores.

Los modelos de extensión permiten a los tenants ampliar el modelo de datos base para adaptarlo a sus particularidades. Son cruciales para el éxito de SaaS ya que permiten configurar las tablas de forma que cada empresa pueda adaptarlas a un diseño similar al de sus aplicaciones on-premises tradicionales.

La personalización del modelo base está relacionado con el enfoque que hayamos seleccionado para implementarlo. Extender los enfoques aislados es sencillo ya que cada tenant tiene sus propias tablas y basta con alterar su estructura; las modificaciones

sólo afectarán a su conjunto de datos y podemos decir que el modelo de extensión está intrínseco en el modelo base. Sin embargo, se ven afectados los costes de infraestructura y mantenimiento. Por el contrario, en entornos compartidos como el de *shared tables,* la complejidad es mucho mayor; las tablas lógicas de cada tenant deben de ser mapeadas (transformadas) en una tabla física común a todos los suscriptores. Esta situación reduce costes de infraestructura y mantenimiento pero sacrifica el rendimiento e incurre en un incremento elevado de esfuerzo de diseño y desarrollo. Las *técnicas de mapeo de esquemas* estudiadas en este trabajo pertenecen por tanto a este nivel ya que permiten extender el modelo de datos base a las necesidades específicas de una determinada organización, para bajos grados de aislamiento.

3 Un modelo de datos MT²

Proponemos un modelo de datos relacional para la capa de datos en arquitecturas MT². Este modelo permite extender los modelos MT mono-target a fin de que puedan soportar ofertas multi-servicio a los suscriptores, así como la escalabilidad en el portfolio de las funcionalidades ofertadas. Basándonos en el marco de referencia propuesto para MT tradicional, consideramos tres modelos en las bases de datos MT²:

- **Modelo administrativo**, que permite gestionar el entorno MT² como sistema y que es el utilizado en el nivel administrativo de la arquitectura.

- **Modelo base**, conteniendo el esquema común a todos los tenants y que es usado en el nivel de instancia de la aplicación (salvo para copias de seguridad y migraciones).

- **Modelo de extensión,** para la personalización del modelo base a las necesidades del cliente. Este nivel es usado también en el nivel de instancia.

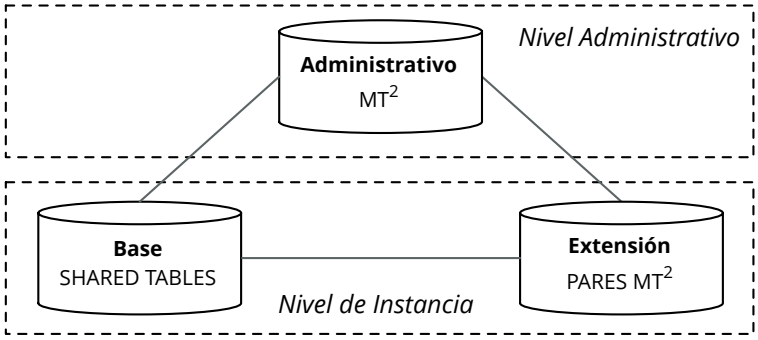


Figura 107. Modelos y enfoques para el modelo de datos MT²

La Figura 107 representa los modelos y enfoques utilizados en esta propuesta, así como el nivel de la arquitectura al que pertenecen. El modelo administrativo no está basado en ningún enfoque y lo llamaremos MT²; al igual que en el caso mono-target no precisa de eficiencia multi-tenant ya que su objetivo es la gestión del sistema MT². Para el modelo base implementamos *multi-tenancy puro* [91] mediante el enfoque de *tablas compartidas* [15], [103]; como modelo de extensión ideamos una nueva técnica de mapeo llamada *Pares MT²*, basada en el enfoque *pares nombre-valor* [103].

3.1 Modelo administrativo

En la Figura 108 recordamos el nivel administrativo de la arquitectura MT^2. Esta vez presentamos en gris y con borde grueso los componentes relacionados con la capa de datos; los nuevos componentes también están destacados con un borde rojo.

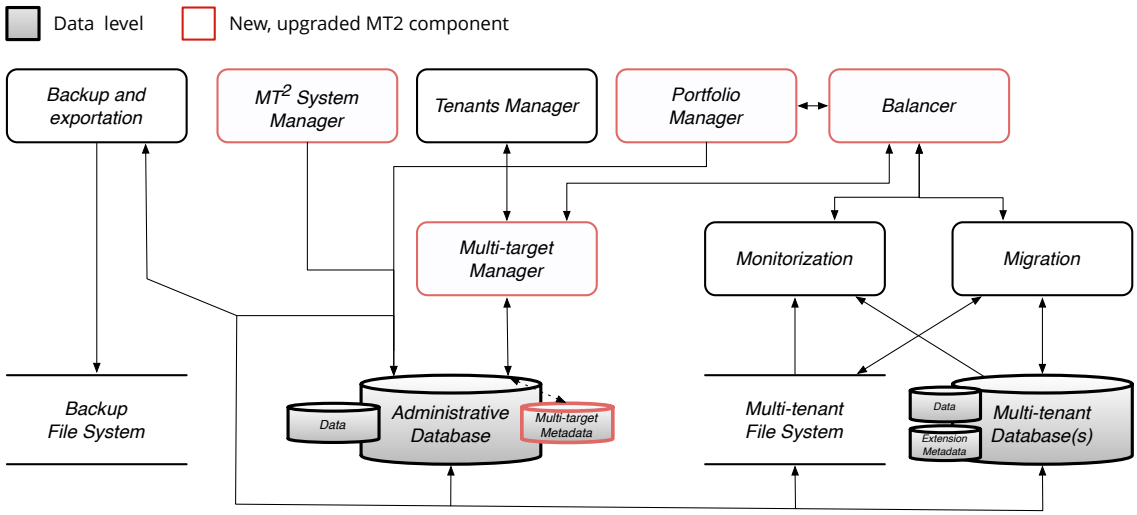


Figura 108. Nivel Administrativo en MT^2 destacando nivel de datos.

Las modificaciones con respecto al modelo tradicional están centradas principalmente en este nivel de gestión. La base de datos administrativa debe dar soporte a nuevos requisitos, como el mantenimiento del portfolio funcional, la suscripción multi-servicio o nuevos retos de privacidad.

La Figura 109 representa el modelo administrativo de bases de datos MT^2. La *Tabla Administrativa de Tenants* (ATT, *Administrative Tenants Table*) almacena la información relacionada con los arrendatarios y no presenta diferencia con respecto al modelo antecesor mono-target. El campo *tenant_id* será usado como clave externa en las tablas de los tenants (TTs) del nivel de instancia.

Las funcionalidades son registradas en la *Tabla Administrativa Funcional* (AFT, *Administrative Functional Table*). Esta tabla representa el portfolio funcional ofrecido por el sistema MT^2. El número de funcionalidades difiere de unos sistemas a otros y puede variar con el tiempo.

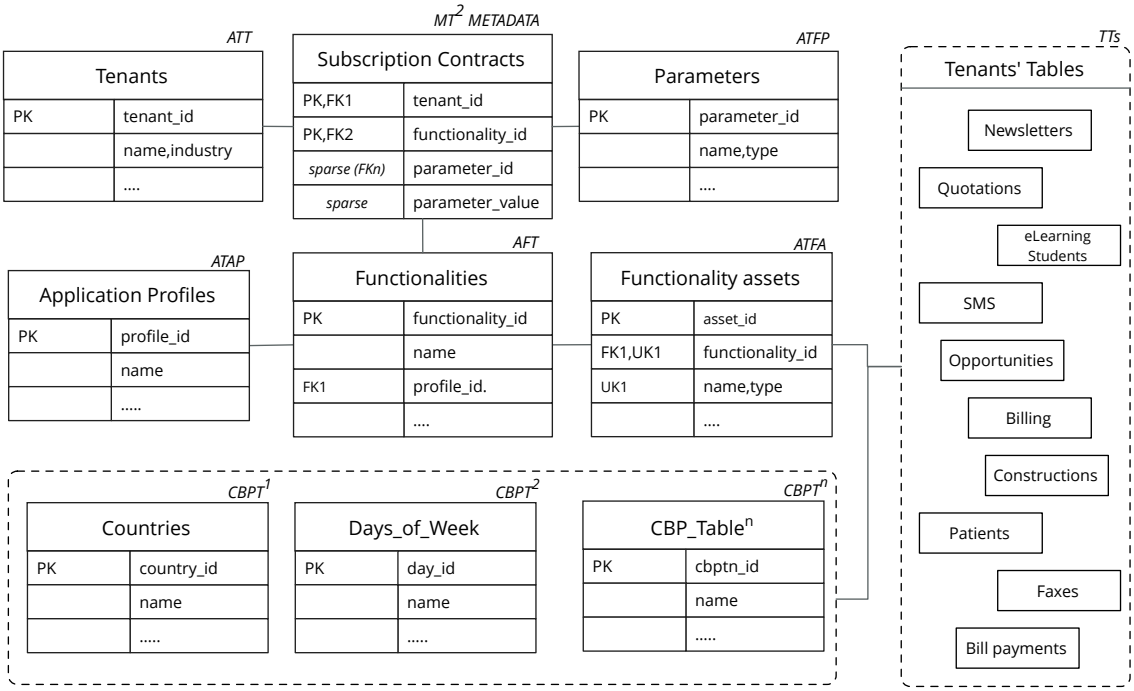


Figura 109. Bases de datos MT². Modelo administrativo

La *Tabla Administrativa de Perfiles de Aplicación* (ATAP, *Administrative Table of Application Profiles)* guarda perfiles de aplicación y su definición. Dos funcionalidades distintas pueden compartir el mismo *perfil*; por ejemplo, SMS y Newsletter pueden ser considerados dentro del perfil CRM, ya que su objetivo principal es la captación de clientes potenciales y la generación de oportunidades de venta; por otro lado, la funcionalidad de facturación podría ser encuadrada en un perfil de ERP. Además de un orden semántico y lógico, la clasificación en perfiles habilita posibilidades analíticas para los vendedores SaaS. La monitorización del sistema nos permite tener estadísticas basadas en esta segmentación, lo cual puede ser de gran interés y ayuda para los proveedores; por ejemplo, las empresas SaaS MT² pueden analizar las ventas o el uso del nivel de instancia no sólo a nivel funcional, sino que también a nivel de perfil de aplicación.

Una única funcionalidad puede comprender varias TTs para almacenar la información de los tenants (tablas o relaciones). La *Tabla Administrativa de Activos de Funcionalidad* (ATFA, *Administrative Table of Functionality Assets)* relaciona las TTs con la AFT; debemos destacar que las relaciones contenidas en la ATFA pueden ser extendidas a cualquier activo software (archivos por ejemplo), pero por simplicidad nos centraremos en TTs. Esta relación debe de ser usada por componentes de seguridad ya

que los tenants que no estén suscritos a una funcionalidad no deben de ser capaces de acceder a sus activos.

En la parte inferior de la figura podemos ver las *Tablas de Negocio Común* (CBTs, *Common Business Tables*). La reutilización de componentes es el pilar fundamental del enfoque MT^2 y, a nivel de base de datos, existen muchas tablas que pueden ser reutilizadas entre tenants independientemente del sector o perfil (e.g., días de la semana, meses, divisas y conversiones). Estas CBTs son almacenadas en el modelo administrativo y sus claves primarias enlazadas como claves externas en las TTs.

Cada suscripción a una funcionalidad tiene sus propias condiciones y estas están reflejadas en los metadatos multi-target de cada tenant. Los *metadatos MT^2* enlazan las cuentas de los tenants con la AFT, controlando las características individuales de esta relación. En la Figura 109, la *Tabla Administrativa de Parámetros Funcionales* (ATFP, *Administrative Table of Functional Parameters*) contiene los parámetros susceptibles de ser parte de un contrato. La clave primaria de ATFP será referenciada como clave externa en los metadatos MT^2 de forma que se instancien los valores específicos de los parámetros en su contrato de suscripción funcional.

Por ejemplo, la suscripción al módulo TPV para los cobros, requiere que se establezca la configuración de conexión proporcionada por el banco. Estos valores de parámetros son individuales de cada tenant y deben ser instanciados en cada caso. Este escenario requiere considerar un número indeterminado de parámetros por funcionalidad, asociado a la tabla de metadatos MT^2; para ello, usaremos el enfoque de *Columnas Dispersas (Sparse Columns)* [113], estudiado en el marco teórico.

La Figura 110 muestra un sistema MT^2 con un portfolio de siete funcionalidades y cuatro posibles parámetros. La ATT contiene 5 organizaciones suscritas a diferentes funcionalidades y en términos distintos. La tabla de metadatos MT^2 registra el contrato de suscripción funcional; tal y como podemos ver en la figura, contiene las claves externas de la ATT y la ATF más una dispersión de cuatro campos (pueden ser más) para articular la instanciación de parámetros funcionales. Las columnas dispersas contienen en los campos impares la clave externa del parámetro funcional e inmediatamente después el valor de dicho parámetro.

ATT

Tenants	
id	name
1	HealthCare 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

Figura 110. Metadatos MT^2. Almacén físico de las suscripciones funcionales de los tenants.

Logical Subscriptions	
Tenant	Functionalities and terms
HealthCare 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*: Multimedia support, XML export - Concrete tracing - *Blog: XML export*

Figura 111. Representación lógicas de los contratos de suscripción funcional (Metadatos MT^2)

La Figura 111 contiene un resumen lógico de las suscripciones funcionales tras la interpretación de la tabla de metadatos MT^2 de la Figura 110. En ella podemos ver cómo el tenant "HealthCare Granada" perteneciente al sector médico, tiene contratadas

las funcionalidades de admisionado de pacientes, facturación y el envío de 500 SMS usando la pasarela gtw1.com.

3.2 Modelo base

La Figura 112 recuerda el nivel de instancia de las AMT^{2}s destacando en gris la capa de datos y con borde rojo los nuevos componentes.

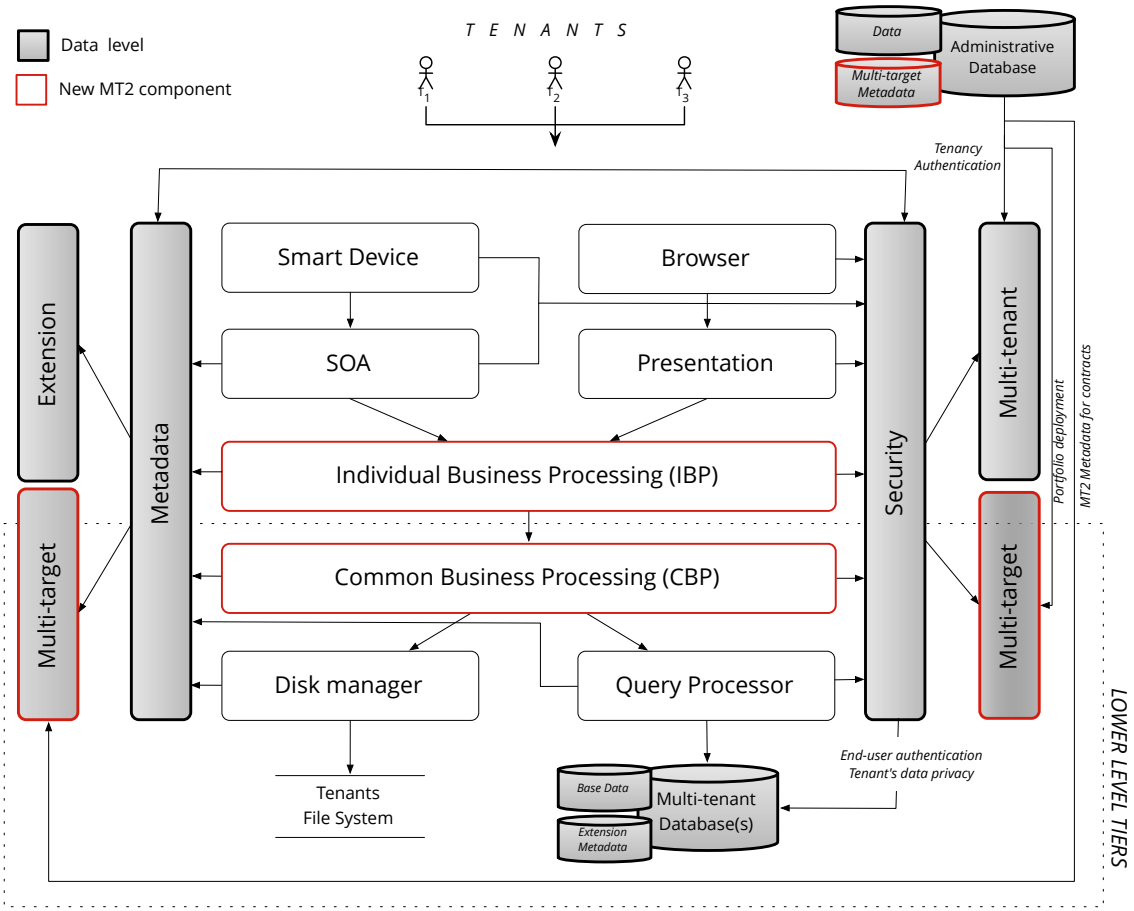


Figura 112. Nivel de instancia en MT^2 destacando nivel de datos.

En este nivel, el modelo base no difiere del modelo tradicional y podemos usar un enfoque mono-target; sin embargo, el carácter multi-funcional del entorno MT^2 requiere consideraciones extras en otros aspectos, como por ejemplo la seguridad funcional.

En nuestro modelo consideramos multi-tenancy puro con el enfoque de *tablas compartidas*; en esta aproximación, los tenants comparten base de datos y tablas por lo que es necesario registrar en cada fila un campo tenant_id como clave externa a la llave primaria de la tabla ATT. De este modo, multi-tenancy se implementa a nivel de

aplicación y no a nivel de sistema usando virtualización con lo que optimizamos la escalabilidad [66]. En la Figura 113 vemos a la izquierda la ATT y la derecha una tabla TT del nivel instancia que implementa este enfoque. En este caso, la TT almacena las *oportunidades de venta* de sus productos que, como podemos intuir por su naturaleza, son de empresas pertenecientes a diferentes sectores. En gris, los registros del tenant 4.

Tenants *ATT*

tenant_id	Name
4	Sur Inmobiliaria
9	Valero Seguros
10	Treve nkk IT Services
...	...

Sales_Opportunities *TT*

tenant_id	record_id	Name	Source	Product
4	1	Alejandro Molina	Internet	Terraced in Marbella
9	2	Fernando Melero	Exhibition Event	Life Insurance
4	3	Joseka Deesign	Direct - Office	Apartment in Naranjos
10	4	John Star	Adwords	Web Design
...	...	...	...	....

Figura 113. Ejemplo de TT del modelo base en MT² con enfoque de tablas compartidas

A diferencia de los entornos mono-target, **no todos los tenants pueden almacenar información en todas las tablas**; el componente de seguridad multi-target debe comprobar los metadatos MT² para ver que el tenant está suscrito a esa funcionalidad.

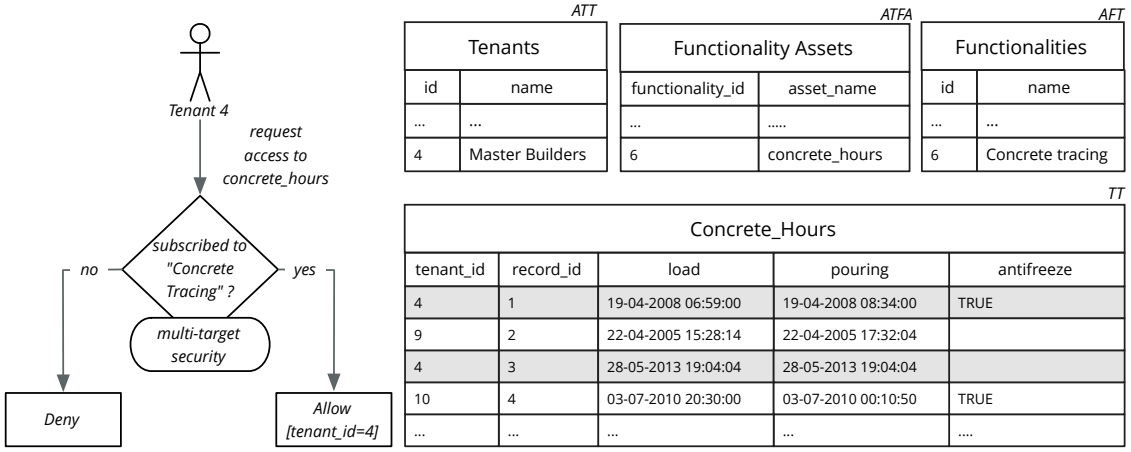


Figura 114. Ejemplo de Tablas Compartidas y comprobación de seguridad MT².

Como ejemplo, en la Figura 114 la tabla *Concrete_Hours* es una TT perteneciente a la funcionalidad *Concrete Tracing* (*Trazabilidad del Hormigón*, propia del sector de la construcción para controlar la calidad del hormigón en ruta). Cuando el Tenant 4 (ver la tabla ATT) intenta acceder a la tabla, los componentes de seguridad MT² necesitan comprobar previamente la tabla de activos de funcionalidad ATFA, para saber que la TT pertenece a la funcionalidad y conceder el permiso. En la parte izquierda de la figura

podemos ver el diagrama de flujo de acceso a la funcionalidad. Tras comprobar vía ATFA y metadatos MT^2 que el tenant está suscrito a Concrete Tracing, la primera parte de la comprobación será positiva. Posteriormente, la privacidad MT tradicional debe asegurar que el tenant lee o actualiza sus registros (marcados en gris) y que además almacena los nuevos con el campo tenant_id=4.

3.3 Modelo de extensión: Pares MT^2

En un entorno compartido y poco aislado, los metadatos de extensión permiten personalizar las tablas base, adaptándolas a las necesidades particulares de los suscriptores. Un usuario final con privilegios suficientes, podrá agregar atributos a las TTs y definir su tipo de datos de modo que los nuevos campos aparecerán sólo al los usuarios finales dentro de la misma tenancy, y no al resto de tenants.

Proponemos *Pares* MT^2 como modelo de extensión de un entorno multi-target; nuestro modelo es una variante al enfoque de tablas de extensión propuesta por Chong (*Pares Nombre-Valor)* [103] y explicado durante el desarrollo del marco teórico. Básicamente, este enfoque se extiende con la información necesaria par dar soporte a un entorno MT^2; la Figura 115 define el modelo con las entidades y sus relaciones.

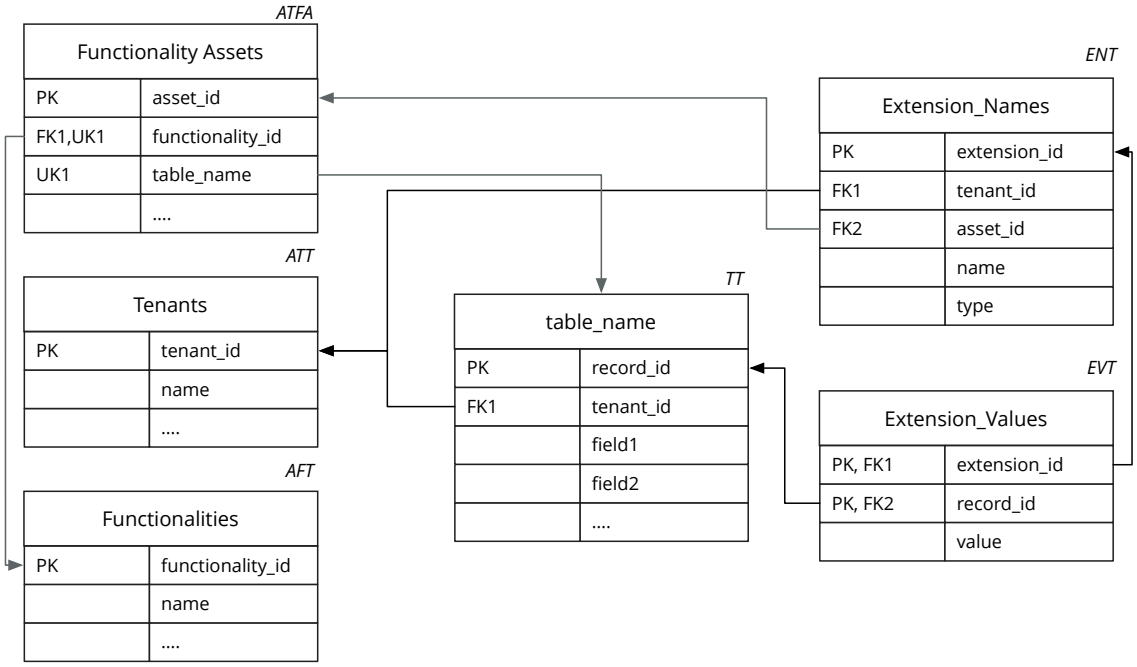


Figura 115. Modelo de extensión Pares MT^2.

En Pares MT^2, la extensibilidad se consigue mediante el uso de dos tablas diferentes; en las *Tablas de Nombre de Extensión* (ENT, *Extension Name Tables*), los tenants definen los campos personalizados de su organización que posteriormente serán almacenados en las *Tablas de Valores de Extensión* (EVT, *Extension Value Tables*).

El campo *value* de la tabla EVT debe de ser flexible (tipo VARCHAR) para almacenar valores de cualquier tipo de los definidos en la tabla ENT. Se podría considerar una variante sobre nuestro enfoque en el que se tuviera una tabla EVT por cada tipo de datos. Esta variante, estudiado en el modelo de extensión de *Tablas Pivotantes*, mejora la eficiencia del sistema, en detrimento del aumento del número de tablas.

Las tablas ENT deben de estar enlazadas a ATFA para saber sobre que tabla física (TT) está definida la extensión. Además, este enlace ENT-ATFA es usado también por los servicios de seguridad MT^2, de forma que prevengan a los tenants de la personalización de TTs relacionadas con funcionalidades no contratadas.

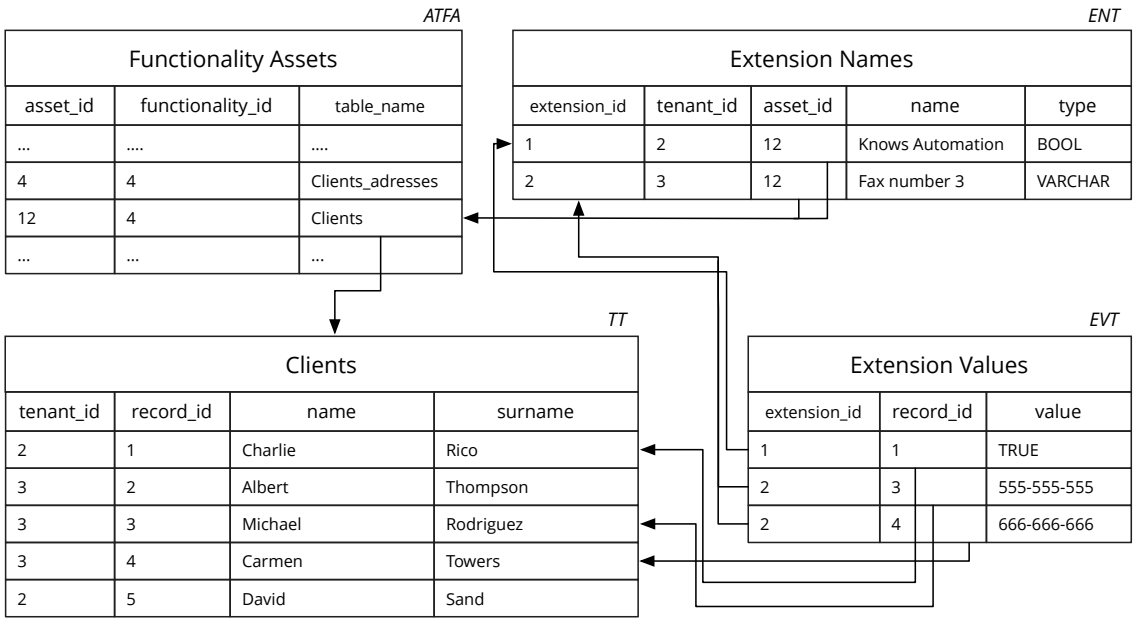


Figura 116. Pares MT^2. Ejemplo de extensión

La Figura 116 nos muestra un ejemplo de este enfoque. Supongamos que los tenants 2 y 3 están suscritos a la funcionalidad *Clients Management* (Gestión de Clientes) con clave primaria *functionality_id* 4. En la tabla ATFA encontramos dos tablas físicas de tenants relacionadas con ella: *Client_addresses* (direcciones de cliente) con identificador asset_id 4 y *Clients* (clientes) con asset_id 12. Esta última tabla de clientes carece de

algunos campos útiles para los tenants como por ejemplo el grado de conocimiento de domótica (*knows automation)* para el tenant 2 y un campo extra de número de fax (*Fax Number 3*) para el tenant 3; por tanto, el modelo base debe de ser extendido. Para ello, la tabla ENT almacena ambas definiciones y las relaciona con las TTs dentro de ATFA de forma que la aplicación sepa que las extensiones son aplicables sobre la tabla base *Clients*. El campo *record_id* determina en las ENV sobre qué registro de la tabla de clientes se aplica la extensión.

El componente transformador de consultas debe de considerar los metadatos de extensión y el enfoque usado para la obtención de registros en las consultas; de esta forma, para cada tenant las tablas lógicas diferirán en las extensiones personalizadas. En la Figura 117 podemos ver las tablas lógicas de cada uno de los tenants.

Clients T_2			
record_id	name	surname	knows automation
1	Charlie	Rico	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	

Figura 117. Ejemplo Pares MT². Tablas lógicas de la tabla física Clientes para los tenants 2 y 3

Capítulo IX

Gestión y soporte MT^2

La elección de MT^2 como arquitectura de aplicación marca unas metas en el proceso de desarrollo que pueden ser complejas de lograr. Previa a la presentación de una implementación real MT^2, este capítulo muestra algunos aspectos prácticos relacionados con la gestión y soporte de estos noveles sistemas. El objetivo es proporcionar una guía en los puntos más relevantes que ayuden al desarrollo de sistemas basados en la propuesta MT^2 presentada.

1 Introducción

MT^2 incorpora numerosas mejoras en las aplicaciones cloud multi-tenant; sin embargo, aquellos objetivos de desarrollo que difieran del modelo tradicional pueden ser de difícil consecución.

El paso de multi-tenant a multi-target es algo nuevo y por tanto no existen referencias y/o guías de desarrollo. Las empresas pueden sentirse desconcertadas al no tener herramientas que marquen una hoja de ruta.

Nuestra propuesta beneficia y da soporte a la agilidad, es decir, alivia la carga de trabajo de los programadores; sin embargo, esta ventaja se consigue una vez desarrollado el entorno, esto es, la plataforma MT^2. Es precisamente a esta fase de desarrollo a la que están destinadas estas secciones.

Este trabajo ya define modelos y marcos de referencia que dan soporte al desarrollo y diseño de estos sistemas; sin embargo, dada la originalidad de la arquitectura, se precisa una exposición de ejemplos menos formales, explicando aspectos prácticos para reforzar el entendimiento.

Así, este capítulo tiene como objetivo eliminar el miedo al trabajo en vano, reduciendo el desconocimiento cuando trabajamos en nuevas materias. Facilitamos recursos que ayuden al soporte y gestión de este tipo de sistemas de modo que las empresas pueden tomar estas nociones como base, sin tener que partir de cero. Por ejemplo, uno de los aspectos claves que distingue las aplicaciones MT^2 es el portfolio funcional pero, ¿cómo mantener ese portfolio?, ¿qué características se han de tener en cuenta en las suscripciones multi-funcionales? Este y otros aspectos serán analizados en las siguientes secciones.

2 Gestión funcional

Los sistemas MT^2 son escalables a nivel funcional y por tanto, el número de servicios ofrecidos así como las condiciones particulares de estos puede variar con el tiempo.

Según veíamos en el capítulo de arquitectura, el *Gestor de Portfolio* es el componente de las AMT^2s encargado de las labores de gestión y soporte de las funcionalidades que despliegue el sistema. El conjunto total de las funcionalidades y sus características son

almacenadas en la base de datos administrativa. Dentro de ella, la *Tabla Administrativa de Funcionalidades (TAF)* registra las funcionalidades presentes en el entorno multi-target (portfolio funcional).

La Figura 118 muestra un ejemplo de portfolio con diez funcionalidades de baja granularidad. Dichas funcionalidades serán asociadas a los tenants por el *Gestor Multi-target* en la configuración de su suscripción.

ID	Nombre
1	SMS
2	Clientes potenciales
3	Control de productos
4	Pacientes
5	Newsletters
6	Citas
7	Encuestas
8	Noticias
9	Admisionado de pacientes
10	Fuentes de clientes

Figura 118. Ejemplo de Portfolio Funcional

Según su naturaleza, algunas funcionalidades precisan tener definidos *parámetros funcionales*, que serán instanciados en los contratos funcionales de cada cliente. Por ejemplo, la funcionalidad SMS deberá tener (al menos) un parámetro por suscripción, que determine el número de mensajes contratados. Otras funcionalidades pueden requerir documentación no estructurada (radiografías de los pacientes por ejemplo) o la capacidad de exportación a otros formatos (PDF, Excel, etc.).

Esta asociación puede ser articulada mediante una relación entre tablas; por ejemplo, si analizamos las relaciones de la Figura 119, obtenemos la traducción lógica en la Figura 120, en la que se detallan los parámetros funcionales asociados a cada funcionalidad.

Funcionalidades

ID	Nombre
1	SMS
2	Pacientes
3	Informes OLAP

Relación

ID_Func	ID_Par
1	1
1	2
1	4
2	1
3	5
3	1

Parámetros funcionales

ID	Nombre
1	Exportación Excel
2	Gateway URL
3	Documentación
4	Máximo Registros
5	Exportación PDF

Figura 119. Asociación de parámetros a funcionalidades en MT²

Funcionalidad	Parámetros
SMS	Exportación excel Pasarela de envío (Gateway URL). Máximo de registros (límite SMS)
Pacientes	Documentación
Informes OLAP	Exportación Excel Exportación PDF

Figura 120. Conjunto de parámetros funcionales por funcionalidad

Las funcionalidades pueden ser muchas y variadas por lo que es conveniente agruparlas en *grupos funcionales*. Un grupo funcional está compuesto por una serie de funcionalidades con un perfil semejante. El *Gestor de Portfolio* perteneciente de Master Panel (nivel administrativo MT²) es el responsable de la creación de grupos funcionales y de definir las funcionalidades de cada grupo.

ID	Nombre
1	SMS
2	Clientes potenciales
3	Control de productos
4	Pacientes
5	Newsletters
6	Citas
7	Encuestas
8	Noticias
9	Admisionado de pacientes
10	Fuentes de clientes

Funcionalidad	Grupo
1	1
2	2
3	3
4	4
5	1
6	2
7	5
8	5
9	4
10	2

ID	Nombre
1	Marketing
2	Clientes (CRM)
3	Almacén
4	Clínica
5	CMS

Figura 121. Asociación de funcionalidades a grupos en MT²

Para modelar esta agrupación establecemos una relación N:1 entre funcionalidades y sus grupos; la Figura 121 muestra un ejemplo de esta relación. Así, el grupo funcional *Clientes (CRM)* estará formado por las funcionalidades: Clientes potenciales, Citas y Fuentes de clientes. En la Figura 122 podemos ver un cuadro resumen con la traducción lógica de este mismo ejemplo.

Grupo	Funcionalidades
Marketing	SMS Newsletters
Clientes (CRM)	Clientes potenciales Citas Fuentes de clientes
Almacén	Control de productos
Clínica	Pacientes Adminisionado de pacientes
CMS	Encuestas Noticias

Figura 122. Ejemplo de grupos funcionales

La agrupación de funcionalidades sirve para:

- **Organización y usabilidad** en el nivel de presentación de datos. Por ejemplo, podemos crear un grupo funcional "Clientes" que agrupe las funcionalidades: "Clientes", "Citas" y "Fuentes de captación". De esta forma los usuarios finales tendrán acceso más usable a las funcionalidades al estar agrupadas.

- **Aplicaciones de marketing** para los vendedores. El vendedor del entorno MT² puede lanzar una campaña que oferte aquellas funcionalidades del grupo que le falten a cada tenant. También puede darse el caso de el desarrollo de una nueva funcionalidad, y que ésta se publicite a todos tenants que despliegan las funcionalidades del mismo grupo.

Una de las características deseables de los sistemas MT² es la **seguridad en el acceso a funcionalidades**. Aún perteneciendo al mismo tenant, todos los usuarios finales no deben poder hacer todo en una funcionalidad; por tanto, los *privilegios* que la

funcionalidad admita deben ser considerados previamente en su definición al incorporarla al portfolio del sistema.

El administrador a nivel de instancia seleccionará un conjunto de privilegios y por tanto establecerá los roles definitivos del usuario final (determinando su *rol funcional*). Los privilegios a seleccionar estarán siempre dentro de un conjunto definido por el Gestor de Portfolio MT^2. Ejemplos típicos de privilegios los podemos ver en la Figura 123.

Privilegio	Descripción
Ver	Ver funcionalidad
Insertar	Insertar registro dentro de la funcionalidad
Editar	Editar registros existentes en la funcionalidad
Eliminar	Eliminarr registros existentes
Asignar	Asignar registro (un cliente potencial a otro usuario, un comercial por ejemplo, un paciente a un médico, etc.).
...	...

Figura 123. Ejemplos de privilegios funcionales

Podemos modelar esta relación de forma análoga a los parámetros funcionales. Como ejemplo mostramos las tablas relacionales de la Figura 124; en este caso las funcionalidades tendrían definidos los privilegios de la Figura 125

Funcionalidades

ID	Nombre
1	SMS
2	Pacientes

Relación

Funcionalidad	Privilegio
1	1
1	2
2	1
2	2
2	3
2	4
2	5

Privilegios

ID	Nombre
1	Ver
2	Insertar
3	Editar
4	Eliminar
5	Asignar

Figura 124. Relación privilegios- funcionalidades

Funcionalidad	Privilegio
SMS	Ver Insertar
Pacientes	Ver Insertar Editar Eliminar Asignar

Figura 125. Conjunto de privilegios por funcionalidad

3 Suscripciones multi-funcionales

Las suscripciones a un sistema MT^2 difieren de las MT tradicionales en que en el contrato es multi-funcional. Esto es, debemos especificar qué funcionalidades desplegar por cada tenant, así como las particularidades específicas de cada una de ellas (si las tuviera).

Básicamente, las suscripciones de los tenants son almacenadas en los *metadatos MT^2*, que contienen el conjunto de funcionalidades contratadas. Una forma sencilla de modelar la relación la tenemos en la Figura 126. A través de la tabla de metadatos MT^2 podemos ver las funcionalidades contratadas por cada uno de los suscriptores. En el ejemplo, si analizamos la suscripción funcional del tenant 3 (ISD Ibérica) vemos que está suscrito a *Control de Productos, Encuestas y Noticias*.

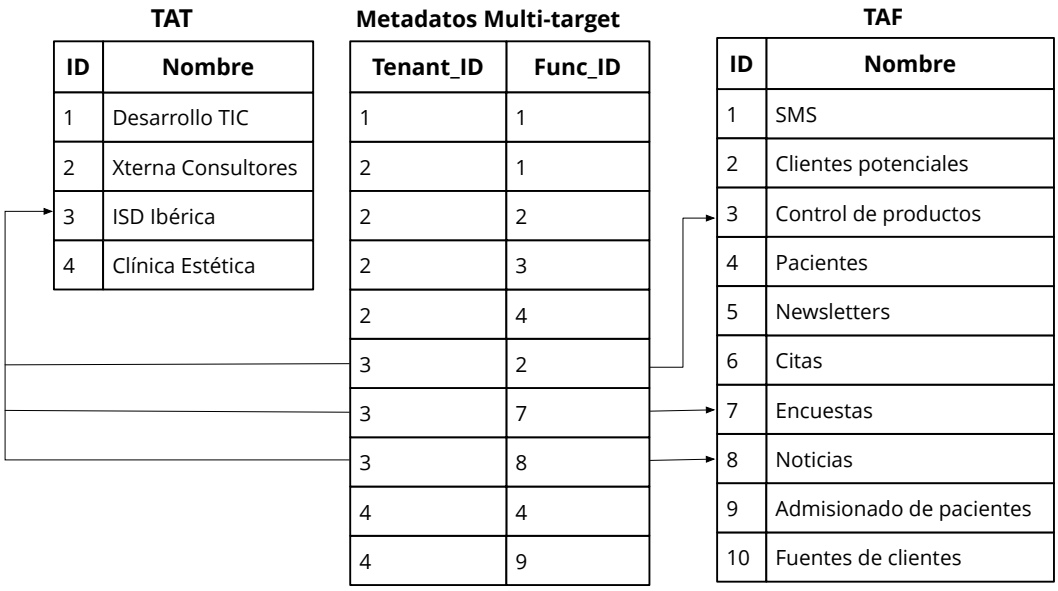


Figura 126. Los metadatos multi-target relacionan las funcionalidades con los clientes

La relación entre ambas entidades puede contener más información además de las llaves primarias de ambas tablas. En ella se pueden dar valores a *parámetros funcionales* definiendo el contrato del cliente. La gestión funcional se encarga de definir estos parámetros funcionales y ahora es el momento de darles valor.

En el capítulo de Bases de datos MT^2 vimos un ejemplo relacionar detallado de cómo llevar a cabo dicha instanciación mediante *columnas dispersas*. Este enfoque consiste en añadir un conjunto disperso limitado de columnas a la derecha de la relación tenant-funcionalidad, conteniendo el identificador del parámetro funcional más su valor en la suscripción.

En la Figura 127 tenemos un ejemplo en el que podemos ver cómo varían las suscripciones de los tenants a la misma funcionalidad. Cada uno de ellos está suscrito a SMS pero tienen contratados 500, 200 y 100 mensajes de texto respectivamente. El valor del parámetro funcional "Máximo Registros" es instanciado en el contrato (metadatos MT^2) y permite al vendedor llevar un registro de los mensajes por consumir de cada organización.

Parámetros funcionales

ID	Nombre
1	Exportación Excel
2	Gateway URL
3	Documentación
4	Máximo Registros
5	...

TAF

ID	Nombre
1	SMS
2	...
3	....

Metadatos MT^2

Tenant_ID	Func_ID	Par_ID	Par_Valor	Par_ID	Par_Valor	...
1	1	4	500	...	...	...
2	1	4	200	...	...	...
3	3	4	100	...	...	...

Figura 127. Metadatos MT^2 con valores de parámetros funcionales

La gestión de suscripciones debe de estar apoyada a su vez por un sistema automatizado de *control de expiraciones*. El éxito de los sistemas MT (multi-target o no) está basado en las economías en escala y por tanto, en la venta a un número elevado de suscriptores.

Este número alto de suscripciones se haría imposible de gobernar, sin los adecuados medios de control de expiración. Este control de expiraciones de suscripción debe disponer de herramientas e información relevantes para una correcta gestión de las renovaciones. Algunos ejemplos son:

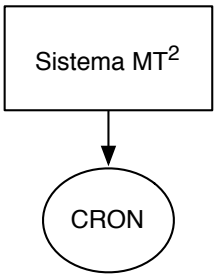
- **Fecha de alta y frecuencia de renovación**. Dicha información está presente en la TAT (Tabla Administrativa de Tenants).
- **Mecanismos** de expiración. El sistema MT2 debe de disponer de herramientas automatizadas que se ejecuten diariamente.
    - Notifiquen a los tenants de la expiración de contrato de suscripción.
    - Proporcionen enlaces a procesos automáticos de renovación y/o modificación de los contratos de suscripción.
    - Suspendan la suscripción en caso de no renovación del cliente.

En la Figura 128 vemos un ejemplo concreto de estructura TAT; en el podemos ver que la empresa *Desarrollo TIC* debe renovar mensualmente el servicio de la versión PRO y que la fecha de alta fue el 19 de Abril del año 2012

ID	Nombre	Dirección	CIF	Servidor_ID	Régimen	F.Alta	Divisa	...
1	Desarrollo TIC	...	...	1	MES	19/04/2012	EUR	...
2	Soluciones PYC	...	..	1	TRI	13/10/2014	USD	...
3	ISD Ibérica	...	...	2	ANUAL	28/05/2013	EUR	...

Figura 128. Ejemplo de TAT con control de expiraciones

Una forma de articular este mecanismo es mediante el demonio CRON de los sistemas UNIX [198]. En la Figura 129 vemos como el sistema ejecuta el script de control de expiraciones diariamente a las 3am.




```
0 3 * * * /root/control_expiracion.sh
```

Figura 129. Control de expiración de suscripciones en MT²

4 Gestión de balanceo

El *control de balanceo* tiene como objetivo preservar el rendimiento del entorno, distribuyendo de forma eficiente los pesos computacionales entre servidores del farm MT². El componente encargado de llevarlo a cabo es el *Gestor de Balanceo* del nivel administrativo MT², que determina en qué servidores alojar (o trasladar) los clientes para equilibrar la carga en la granja.

A diferencia de entornos MT tradicionales, el enfoque multi-funcional puede dotar al proveedor del servicio de **nuevas posibilidades de gestión** dado el carácter multi-sectorial de sus clientes. Por ejemplo, se puede plantear el balanceo de carga ponderando el sector y perfil de los tenants para tener un portfolio más vertical por servidor.

En principio, lo más lógico es balancear la granja sin tener en cuenta el sector, sino meramente la carga computacional de cada uno de los suscriptores. En este sentido, los propietarios de los sistemas MT² pueden disponer de una ventaja frente al modelo tradicional al poder usar la *estimación de carga computacional* de las funcionalidades. Aunque pueda estimarse por la complejidad de las operaciones que implementa, lo ideal es obtener el nivel de carga computacional de forma experimental, como fruto de estudios y pruebas de carga computacional. Así, en la Figura 130, vemos como la TAF contiene esta información, que es establecida por el Gestor de Portfolio.

ID	Nombre	Carga
1	Clientes potenciales	6
2	Control de presencia	5
3	CMS de Noticias	4
4	Informes OLAP	9

Figura 130. TAF con información de carga

Por tanto, existen funcionalidades que por su propia naturaleza supondrán un mayor gasto o computacional o de almacenamiento. Por ejemplo, un módulo de informes estadísticos necesitará mayor uso de CPU que otro destinado al simple de registro de clientes. Un módulo CMS necesitará mayor espacio en disco por tener que almacenar los documentos de descarga. Teniendo en cuenta estos parámetros, no sería lógico por tanto agrupar todos los tenants con altas necesidades computacionales o de disco en el mismo servidor. Por ello, el Gestor de Balanceo debe comunicarse con el Gestor Multi-target para examinar las funcionalidades de cada cliente y determinar el servidor de alojamiento.

Funcionalidades

ID	Nombre	Carga
1	Clientes potenciales	6
2	Control de presencia	5
3	CMS de Noticias	4
4	Informes OLAP Clientes	9

Metadatos Multi-target

Tenant_ID	Func_ID
1	1
1	4
2	2
2	3
3	2
5	3
6	2

Servidores

ID	Tipo
1	Amazon EC2 AMI
2	IBM QC 2 GHz
3	IBM QC 2,2GHz
4	Amazon EC2 CentOS

Tenants

ID	Nombre	Server_ID
1	Desarrollo TIC	1
2	Xterna Consultores	1
3	ISD Ibérica	1
4	Clínica Estética	1
5	Zinia	2
6	Castro Abogados	3
...	...	....

Figura 131. TAF con estimación de carga y Metadatos Multi-target

Teniendo esta información, en cualquier momento podemos determinar la carga individual de cada cliente y por ende, de un servidor. Por ejemplo, si relacionamos los datos de las tablas presentes en la Figura 131, la carga total de los servidores vendrá determinada por la suma de las cargas individuales de cliente. En la Figura 132 obtenemos la carga total estimada del servidor S1.

Tenant_ID	Carga
1	15
2	9
3	5
Suma S1	**29**

Figura 132. Servidor 1: Carga total por suma individual de cada tenant

La *migración* no debe tener en cuenta únicamente el nivel de carga estimado del servidor, sino que también se debe ponderar el volumen de actividad reportado por el gestor de monitorización. Si esta ponderación sobrepasa un determinado *umbral*, el gestor de migración procederá automáticamente al traslado de cuentas. Por cada servidor, el sistema almacenará los datos de umbral y carga actual (Figura 133).

Server_ID	Umbral	Carga Actual
1	25	**29**
2	10	9
3	30	24
4	15	15

Figura 133. Cálculo dinámico de carga

La cuenta o cuentas que se migren no tienen por qué ser las más pesadas, sino de aquellas que minimicen la carga de cada servidor. Considerando la información que dispone, el gestor de balanceo debe implementar una lógica de migración que logre equilibrio de carga en toda la granja. En la Figura 134 la lógica de migración resulta en el traslado de la cuenta de cliente 3 con una carga de 5 al servidor número 3, dando lugar a nueva situación de equilibrio bajo umbral. De este modo, el sistema

permanecerá estable dentro de los parámetros definidos en los metadatos administrativos.

Server_ID	Umbral	Carga Actual
1	25	**24**
2	10	9
3	30	**29**
4	15	15

Figura 134. El sistema MT² se estabiliza tras la migración del cliente

El volumen de actividad y necesidades de un cliente puede variar dinámicamente, bien por un aumento de uso, bien por un aumento del contrato funcional. En cualquier caso, el Gestor de Balanceo debe ejecutar de manera sistemática el proceso de comprobación de carga ya que el sistema puede verse comprometido en cualquier momento disminuyendo el rendimiento.

5 Gestión de disco

Los usuarios finales deben poder almacenar información no estructurada (archivos) relacionándola con registros de funcionalidades presentes en la base de datos. Los ficheros (imágenes, videos, documentos, etc.) están guardados físicamente en disco, pero precisan de tablas en la BDT para almacenar los metadatos de los archivos y así poder recuperarlos.

A diferencia de MT, en MT² la tabla de documentación debe de llevar un registro de la funcionalidad a la cual pertenece el archivo, como medida de seguridad y privacidad del sistema.

La Figura 135 representa las relaciones entre la tabla de documentación, la TAF, y una tabla de la BDT correspondiente a una funcionalidad. El campo *Func_ID* permite identificar la tabla dentro de la BDT, mientras que el campo *Registro_ID* determinará el registro dentro de esa tabla al que pertenece la documentación.

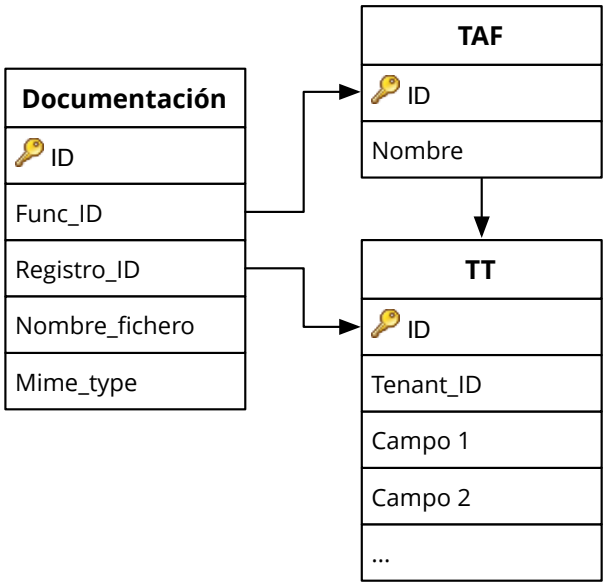


Figura 135. Estructura de la tabla de documentación

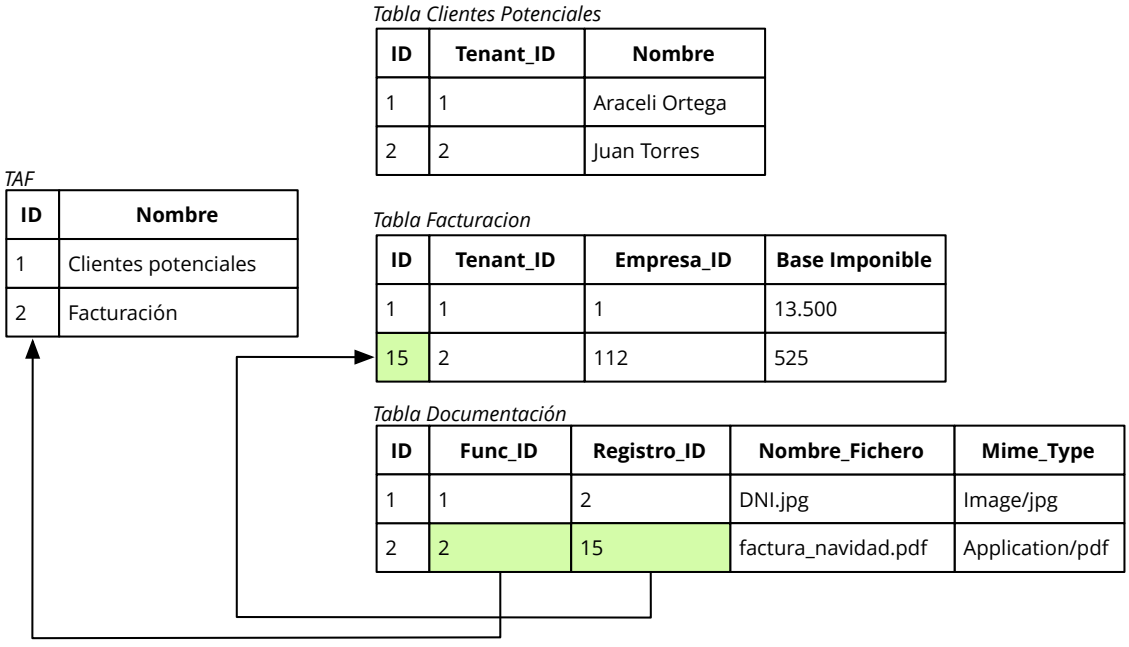


Figura 136. Ejemplo de identificación de documentación en el sistema

En la Figura 136 vemos un **ejemplo** que nos ilustra en este diseño. Si atendemos al registro con identificador 2 de la tabla, la clave foránea *Func_ID* nos indica que la funcionalidad asociada al registro es la de *Facturación* mientras que *Registro_ID* nos indica el registro propietario dentro de esta tabla, esto es, el registro con identificador 15 dentro de la tabla facturación (factura de 525€ de base imponible). Si hacemos lo mismo para el registro 1, veremos como siguiendo las relaciones descubrimos que el propietario es el cliente potencial *Araceli Ortega.* Destacar que en la tabla

documentación no es necesario almacenar el *Tenant_ID* puesto que el *Registro_ID* ya es único en su tabla de la BDT (y ésta si lo contiene) y no puede darse duplicidad.

Para *almacenar* físicamente el archivo, bastará con que utilicemos como nombre el identificador clave de la tabla. Si se quiere (por mantener orden), se puede crear un directorio por cada tenant y dentro de este toda la documentación del mismo, aunque no es necesario. En ejemplo anterior, quedarían de la siguiente forma considerando una separación de directorios por tenant:

/path/to/documentación/[1]/1.file

/path/to/documentación/[2]/2.file

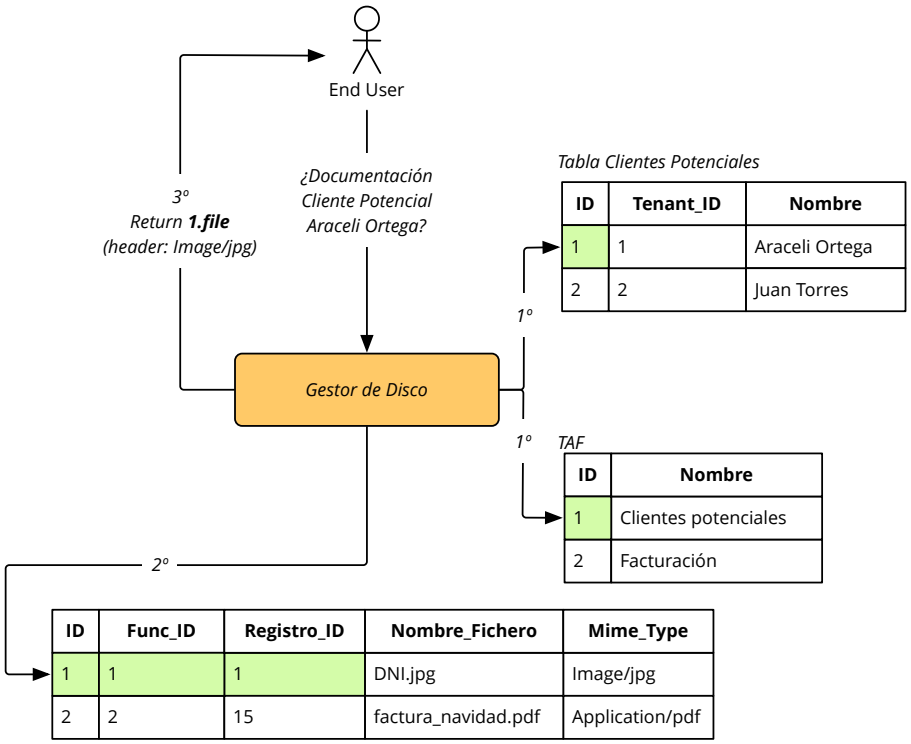


Figura 137. El gestor de disco devuelve al usuario final los archivos solicitados

El gestor de disco podrá *recuperar* un archivo partiendo únicamente del valor ID dentro de la tabla de documentación. La Figura 137 muestra un **ejemplo** en el que un usuario solicita la información no estructurada de un cliente potencial. El usuario final pide al Gestor de Disco (GD) los archivos almacenados en el sistema del cliente potencial *Araceli Ortega.* El GD obtiene el *Func_ID* y el *Registro_ID* de la TAF y la tabla de clientes potenciales respectivamente (1er paso). La dupla (1,1) nos identifica el registro

de la tabla documentación como perteneciente al cliente potencial solicitado (2º paso). Finalmente, el gestor de disco obtiene el fichero 1.file del disco y lo devuelve al usuario como archivo de imagen (ya que tiene un tipo Mime *Image/jpg*) (3er paso)

6 Autenticación y acceso al sistema

En un entorno MT^2, a la pareja tradicional de credenciales (usuario-contraseña) debemos añadirle la identificación de arrendatario. *Tenant_ID* es la clave dentro de la TAT pero al ser un campo numérico puede ser difícil de recordar. En su lugar, podemos usar una cadena identificativa, TENANT_CAD (única para cada tenant), más fácil de memorizar. Por tanto, es deseable que la TAT contenga ambos campos, por motivos de experiencia de usuario. La Figura 138 muestra un ejemplo de TAT y su relación con la tabla de usuarios finales. El proceso de acceso al sistema debe contar con los siguientes datos de identificación: nombre de usuario, contraseña y Tenant_ID.

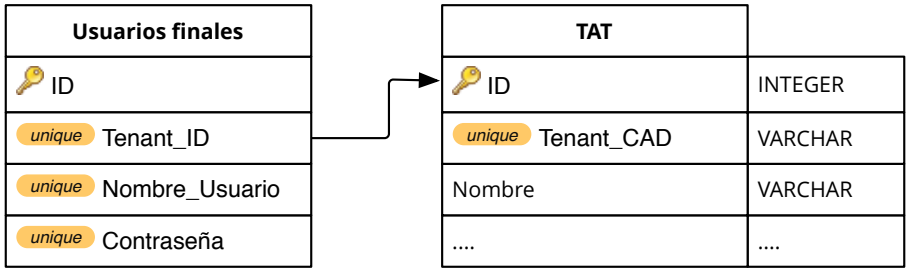


Figura 138. Tablas de usuario final y TAT. Campos usados para credenciales de usuario

Así, la web de acceso al sistema puede ser común a todos los tenants teniendo el aspecto de la Figura 139. En ella vemos como existe una URL de acceso común (https://aplicationMT2.com/acceso/comun) y en la que se especifica por usuario final el Tenant_CAD de cada arrendatario.

Podemos proponer una segunda forma (Figura 139) en la que el *Identificador Tenant* (TENANT_CAD o TENANT_ID) esté implícito en la cadena URL. De esta forma, cada tenant tiene su propia URL, pudiendo personalizar la web de acceso al arrendatario en cuestión. En esta variante, el sistema considerará el identificador como un valor fijo, y validará las credenciales dentro del conjunto de usuarios finales que pertenezcan a ese arrendatario.

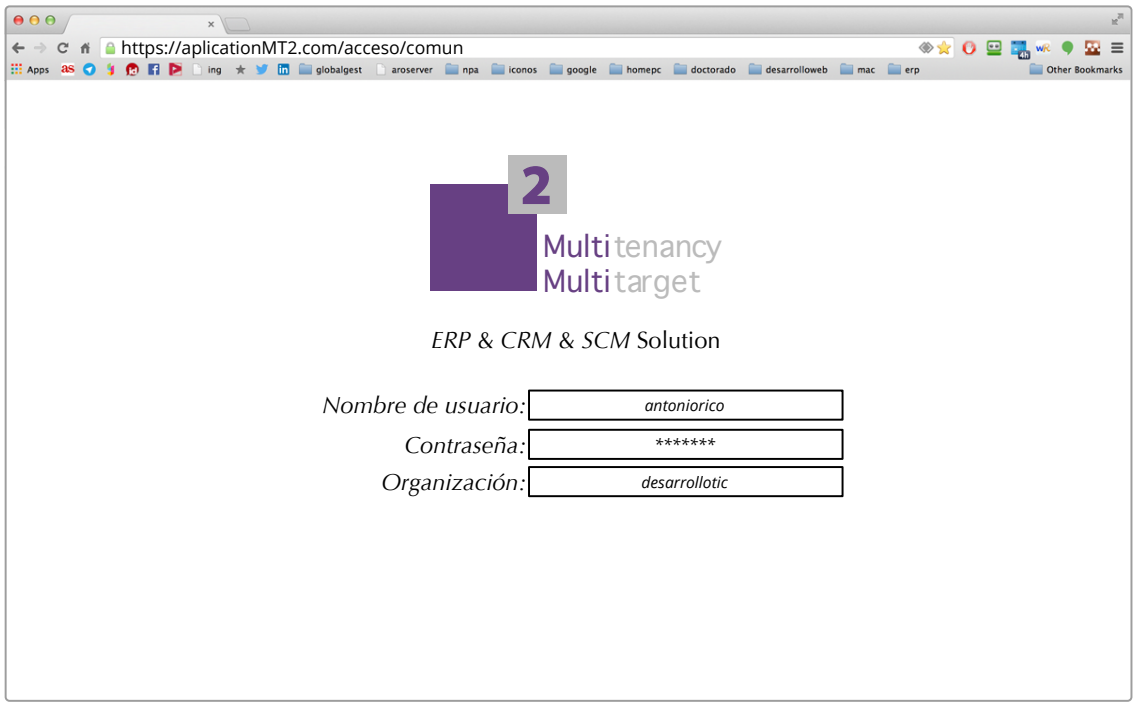


Figura 139. Acceso al sistema MT²: URL común

Figura 140. Acceso al sistema MT²: URL personalizada con TENANT_CAD desarrollotic

7 Niveles de usuario final

Del mismo modo que tenemos un nivel administrativo y otro de instancia en una AMT^2, a nivel de usuario final también debemos considerar esta distinción. Ambos podemos verlos representados en la Figura 141 y son los siguientes:

- Usuario de nivel administrativo (UNA)
- Usuario de nivel instancia (UNI)

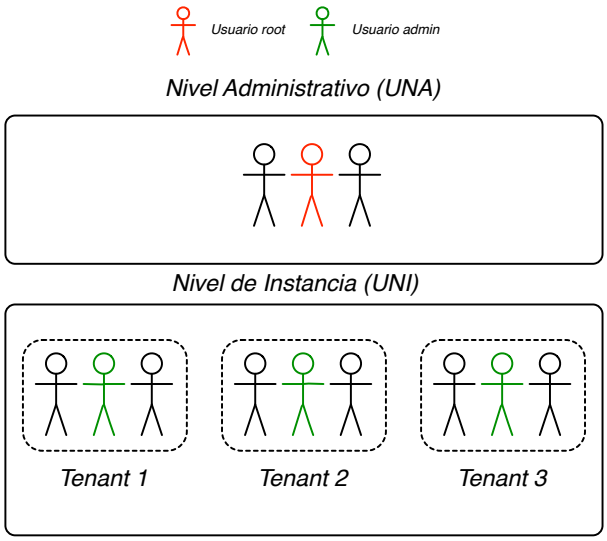


Figura 141. Niveles de usuario en MT^2

El usuario UNA tiene permisos de acceso al framework administrativo (FA) del entorno MT^2 y por ello está capacitado para las efectuar operaciones propias de la administración del sistema tales como:

- Gestión de tenants y suscripciones.
- Control de balanceo de servidores MT.
- Gestión funcional.
- …

El usuario UNI accede al nivel de instancia del entorno y su operativa depende de las características de suscripción definida por algún usuario UNA en el nivel administrativo. Dentro de este nivel UNI, debemos considerar un usuario raíz a partir del cual crear el resto de usuarios; a este usuario lo llamaremos *admin*. Existe un

usuario admin por cada tenant y tiene privilegios para dar de alta otros usuarios finales. Estos usuarios definidos por admin, pertenecerán siempre al mismo tenant y podrán ser dados de alta como administradores. Todos los usuarios de un tenant podrán ser eliminados a excepción del usuario admin.

Del mismo modo que existe un usuario *admin* en el nivel de instancia, debe también un usuario raíz en el nivel administrativo. Este usuario, lo llamaremos *root* y dispone de privilegios totales en el sistema. El usuario root podrá dar de alta nuevos usuarios en el nivel UNA (o usuarios UNA con permisos de creación de nuevos UNA).

Una forma de plasmar esta distinción es **considerar al nivel administrativo como una cuenta de tenant** con acceso a la gestión del entorno MT^2. Las facetas propias del FA las definiremos en el sistema como funcionalidades dentro del portfolio y la suscripción del tenant tendrá por tanto acceso a las mismas. Los usuarios finales de esta tenancy podrán, entre otras, dar de alta nuevas suscripciones en el sistema. Con este enfoque podemos tener varios tenants con acceso al nivel administrativo articulando así **cuentas *reseller*** que permitan a los vendedores de software distribuir el producto.

Veamos un **ejemplo** de cómo un sistema MT^2 puede crear cuentas con acceso a funcionalidades administrativas. En la Figura 142 mostramos la tabla de arrendatarios y de funcionalidades; los registros marcados con fondo rosa están relacionados con el nivel administrativo y son identificados mediante el campo bool FA.

TAT

ID	Tenant
1	Vendedor MT2
2	Desarrollo TIC
3	ACME Abogados
4	Reseller

TAF

ID	Nombre	FA
1	Gestión de tenants	1
2	Balanceo MT	1
3	Gestión funcional	1
4	Clientes potenciales	0
5	CMS Website	0
6	Gestión Resellers	1
7	Facturación	0

Figura 142. Tablas TAT y TAF habilitando el reselling del sistema

Recordamos que los metadatos MT^2 establecen una relación de los servicios contratados por los arrendatarios. Para habilitar el *reselling*, podemos dar de alta en portfolio el nivel administrativo como una funcionalidad más. Según la Figura 143, tanto el tenant 1 como el 4 tendrán acceso a niveles administrativos ya que tienen contratadas funcionalidades de acceso al FA.

Metadatos Multi-target

Tenant_ID	Func_ID
1	1
1	2
1	3
1	6
2	4
2	7
3	7
4	1

Figura 143. Metadatos MT^2. Relación con funcionalidades del nivel administrativo

Así, para dar permisos de acceso a funcionalidades del nivel administrativo, bastará con codificar los privilegios en las tablas de usuarios finales. En la Figura 144 vemos un ejemplo de ello; tanto el usuario "Administrador MT^2", "Antonio Rico" y "Julio Lamolda" tienen permiso de acceso al nivel administrativo.

User_ID	Tenant_ID	Nombre
1	1	Administrador MT2
2	1	Antonio Rico
3	2	Carlos Torres
4	4	Julio Lamolda
5	3	Manuel Sierra
6	3	Pepa Torres
7	2	Tomás Martínez
8	3	Ciriaco Castro

Figura 144. Tabla de usuarios finales. Concesión de privilegios administrativos

8 Servicios web y aplicaciones de terceros

El acceso de aplicaciones de terceros lo conseguimos ofreciendo **servicios web**, cuya llamada permita el desarrollo de nuevas aplicaciones. Esta llamada podrá ser realizada por ejemplo con SOAP para obtener los resultados en el cliente de la aplicación (Figura 145).

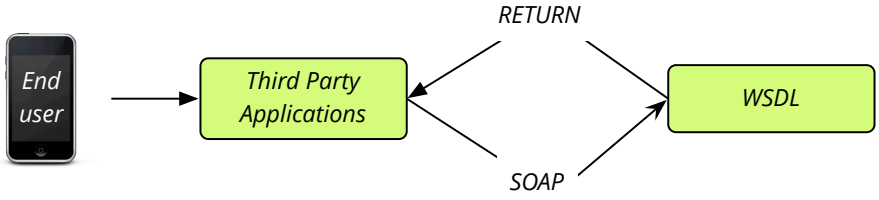


Figura 145. Acceso third party al sistema mediante llamada SOAP al servicio web

Las *Third Party Applications*, deberán tener en cuenta en sus llamadas el entorno MT^2. Por tanto, llamadas inadecuadas producirán errores de violación de acceso a funcionalidades no contratadas por el tenant. La autenticación en web service debe comprender los siguientes campos alojados en el nivel administrativo (Figura 146):

- Identificador de Tenant: TENANT_ID o TENANT_CAD
- Clave de acceso a la API: API_KEY

Campo	Tipo
TENANT_ID	Integer
TENANT_CAD	Varchar *unique*
API_KEY	Varchar
NOMBRE	Varchar

Figura 146. Campos de la TAT relacionados

De este modo, en el conjunto de funciones ofrecidas por la API, existirá una de autenticación semejante a este patrón:

```
(string session_id)conecta_mt2(string tenant_cad, string api_key)
```

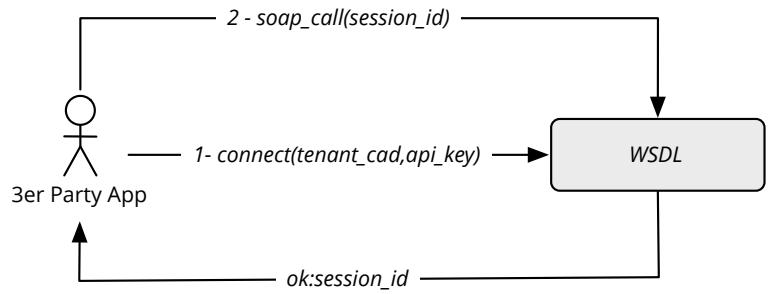


Figura 147. Obtención de identificador de sesión a través de una aplicación de terceros

Tal y como vemos en la Figura 147, la primera llamada de conexión debe ser a la función *connect* que devuelve una cadena con el identificador de sesión (*session_id*). Esta cadena será utilizada en el resto de llamadas como método de autenticación en el sistema. El identificador de sesión *session_id* relacionará las llamadas con la cuenta del tenant y por tanto, evitará accesos que impliquen una violación de las condiciones del servicio.

La existencia de una interfaz a través de la cual podamos acceder a la BDT es **vital en materia de privacidad**. Supongamos por ejemplo el caso de una extranet de clientes para la descarga de facturas o una web de comercio electrónico. Dar credenciales de acceso totales para un *frontend* específico de tenant es algo impensable, ya que cualquier usuario podría recuperar datos de otros arrendatarios. Por otro lado, si el cliente no tiene contratadas funcionalidades relacionadas con el comercio electrónico (carrito de la compra, pedidos, etc.), no debe de poder tener acceso a las mismas mediante llamadas a la API.

El acceso desde aplicaciones terceras debe realizarse mediante una API de servicios web; en primer lugar para asegurar la privacidad de datos entre tenants, y en segundo evitar el acceso a funcionalidades no contratadas.

Capítulo X

Globalgest, un software MT^2

Los capítulos anteriores se centran en formalizar la propuesta detallando la arquitectura, la capa de datos y presentando algunos aspectos prácticos relacionados con gestión y soporte. En este capítulo validamos la aplicabilidad del enfoque presentando *Globalgest*, un software comercial con arquitectura MT^2.

1 Introducción

Globalgest [199] es una aplicación orientada al negocio con arquitectura MT^2 que está actualmente dando servicio a casi 40 empresas de 17 industrias. Comercializado como SaaS, es capaz de desplegar selectivamente más de 200 funcionalidades destinadas a sectores diferentes y/o con niveles de complejidad distinta.

El sistema es escalable mediante funcionalidades que se programan bajo demanda para el cliente. Cada nuevo desarrollo se incorpora a la cartera funcional de Globalgest, pudiendo ser ofertada al resto de suscriptores del sistema.

En Globalgest, el nivel administrativo establece para cada tenant qué funcionalidades dentro del total del portfolio ejecutar en su nivel de instancia. De este modo, configura así su contrato funcional y establece un tipo particular de SIE específico y a medida para cada cliente (ver Figura 148). Así, esta aplicación permite tener a varios clientes con requerimientos software dispares alojados en un mismo sistema.

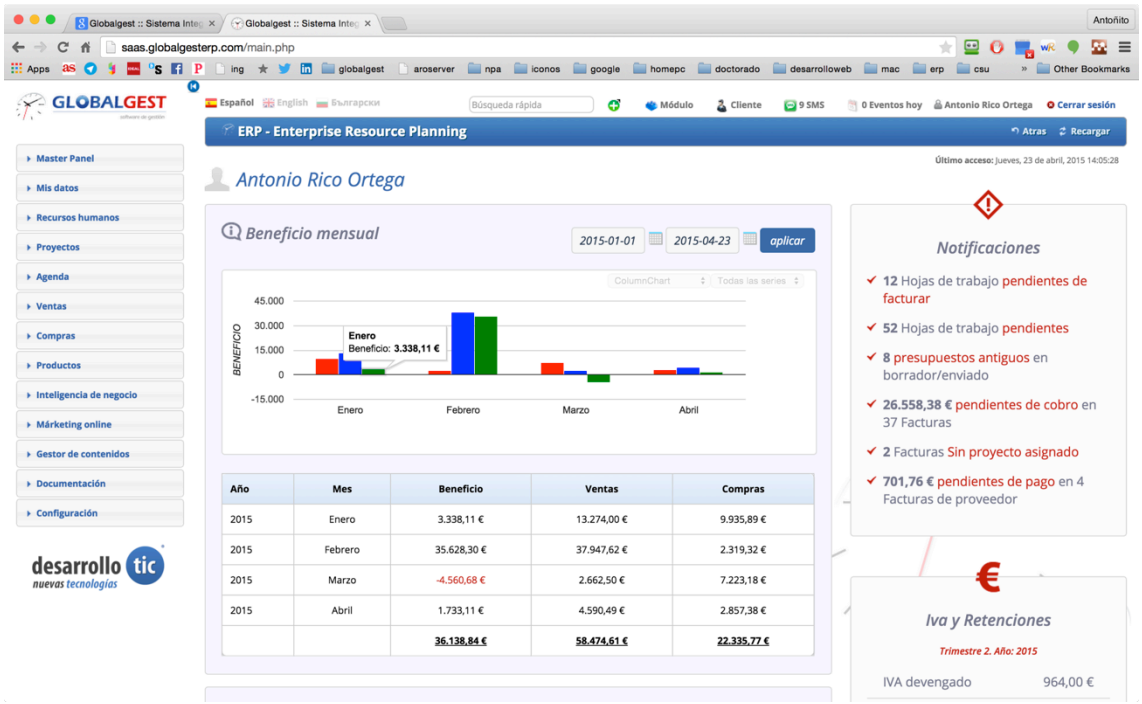


Figura 148. Nivel de Instancia particular de Globalgest para una empresa.

Globalgest existe desde el año 2008, pero no ha sido MT^2 hasta ahora. Los dueños y desarrolladores de la aplicación [200] han mantenido la marca comercial pero variado internamente su arquitectura con los años. La Figura 149 representa la evolución del

sistema con los años; inicialmente, la aplicación se gestó para implantaciones multi-instancia para posteriormente dar paso a una centralización multi-tenant tradicional, con servicio mono-funcional. Hoy en día, gracias al proceso de investigación de este trabajo, el sistema incorpora una capacidad multi-tenant multi-funcional (multi-target).

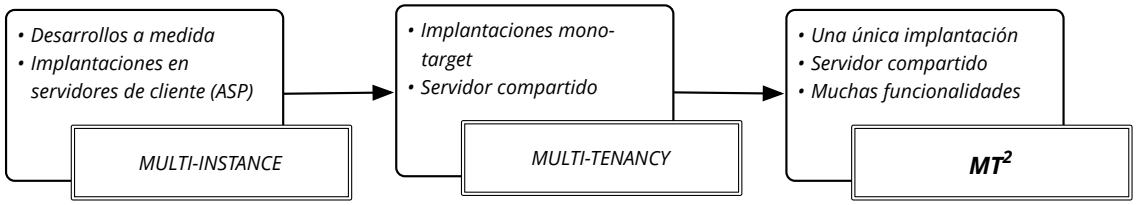


Figura 149. Pasos dados en Globalgest hasta MT²

Globalgest da servicio a pymes (sobre todo a micro empresas) de diferentes industrias y/o con distinta complejidad funcional. La empresa propietaria es Desarrollo TIC [200], una micro empresa con menos de diez trabajadores que, a pesar de tener los recursos propios de su tamaño, aprovecha al máximo la arquitectura MT² subyacente de Globalgest para lograr dar buen servicio, mantener el software y aumentar el espectro de clientes potenciales accediendo a un mercado multi-target. Se puede confirmar que Globalgest es el primer y (de momento) único software MT² del mercado.

2 Nivel de madurez SaaS

Globalgest se encuentra en el *nivel 3 de madurez* según el esquema de Chong. En efecto, en Globalgest existe una única instancia de la aplicación que comparten los diferentes tenants y en donde la capa de seguridad vela por la privacidad de los datos evitando accesos indeseados. Al haber una única instancia del software por implantación, no existe balanceo de carga.

La diferencia con respecto a la clasificación tradicional es la naturaleza multi-target del sistema. Gracias a ella, la aplicación permite dar servicio a organizaciones de diferentes sectores debido a que el sistema es capaz de gestionar y ofertar múltiples funcionalidades dentro su portfolio. En el ejemplo de Figura 150, vemos una única implantación de Globalgest que es capaz de ser utilizado por cuatro organizaciones. En este caso Globalgest, despliega funcionalidades que combinadas dan servicio y configuran un sistema de información personalizado para organizaciones pertenecientes a tres industrias distintas: informática, construcción y sanitaria.

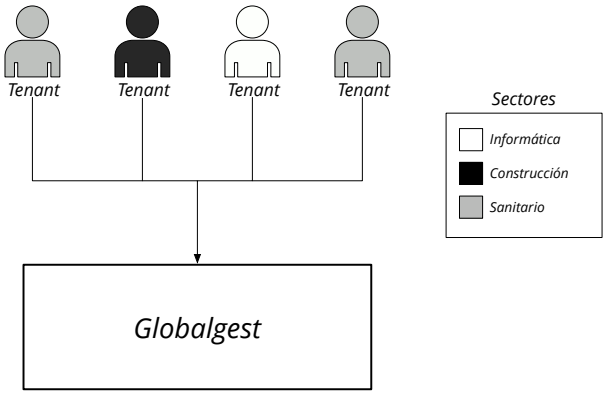


Figura 150. Globalgest es un software SaaS con nivel 3 de madurez

3 Arquitectura

Globalgest es una aplicación MT^2 y como tal, su arquitectura presenta dos grandes niveles: nivel administrativo y nivel de instancia. Recordemos que el nivel administrativo representa el entorno de configuración de la aplicación Globalgest MT^2 sistema (gestor de funcionalidades, tenants, suscripciones, etc.) mientras que el nivel de instancia se corresponde con las aplicaciones que los tenants ejecutan en función del contrato establecido en el nivel administrativo.

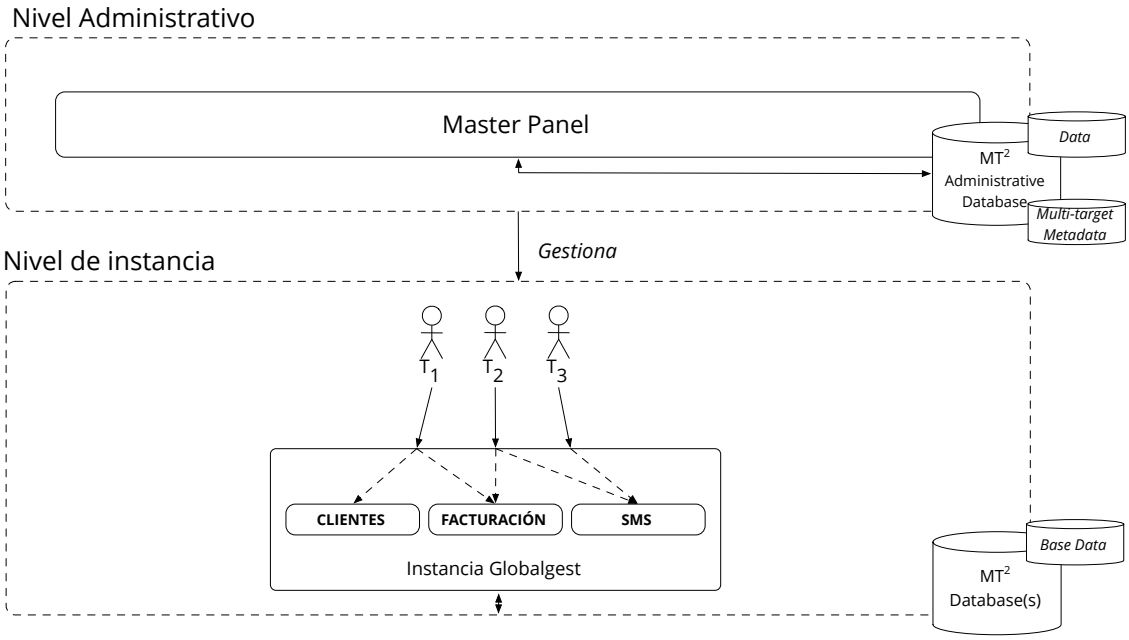


Figura 151. Globalgest: Arquitectura de alto nivel

En la Figura 151 podemos ver representados estos dos niveles. En ella, el sistema despliega 3 funcionalidades de forma selectiva y da servicio a 3 tenants. El primero de

ellos (T_1) tiene contratada la gestión de clientes y la facturación, T_2 está suscrito a la facturación y al envío de mensajes SMS, mientras que T_3 tan sólo tiene contratada la funcionalidad de envío de mensajes de texto.

El nivel de madurez SaaS implica que Globalgest no contempla multi-tenancy farm. Por tanto, a diferencia del modelo general MT^2, su arquitectura carece de componentes de control de balanceo de carga ya que existe una única instancia de aplicación. De momento, la capa de datos tampoco contempla un modelo de extensión pero dicha carencia es solventada con la programación de nuevas funcionalidades adaptadas a las necesidades de cliente. A pesar de ello, es intención de los propietarios del sistema incluir la extensión de datos en futuras actualizaciones. Más adelante estudiaremos con más detalle el diseño y modelado funcional.

4 Nivel administrativo

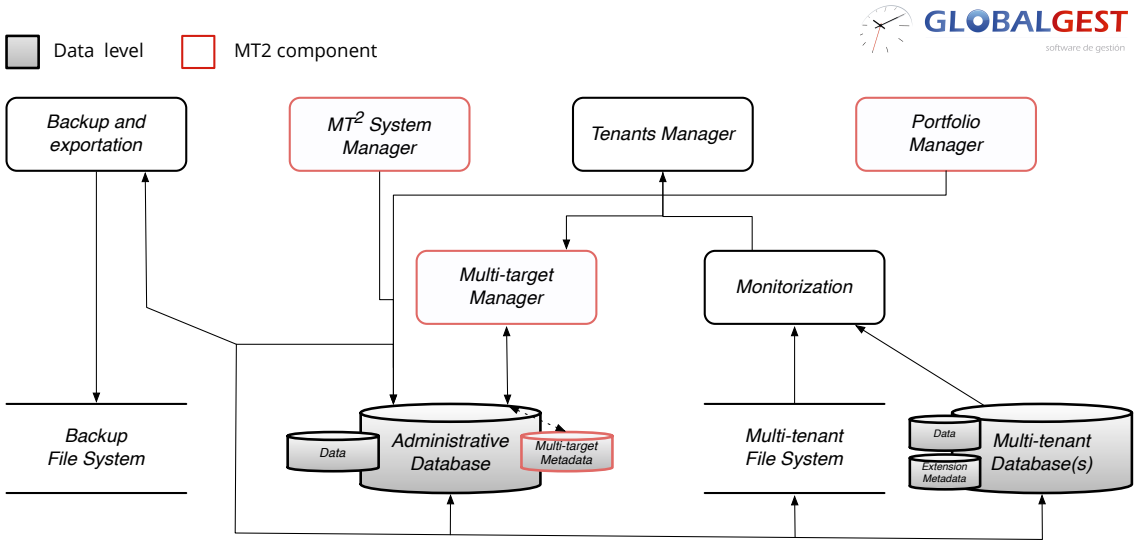


Figura 152. Globalgest: Arquitectura del nivel administrativo

El nivel administrativo en Globalgest recibe el nombre de Master Panel y se muestra únicamente al los usuarios administradores. La Figura 152 se corresponde con la arquitectura del sistema en este nivel. Al no tener un farm de instancias, carece de componentes de balanceo de carga. A pesar de esto, si que hay un componente de monitorización que registra y mide el volumen de accesos de cada uno de los tenants. Este fue programado para medir y demostrar las ventajas de los sistemas MT^2 y no por motivos de balanceo.

A nivel de presentación, el Master Panel se representa como un grupo funcional más que comprende una serie de módulos.

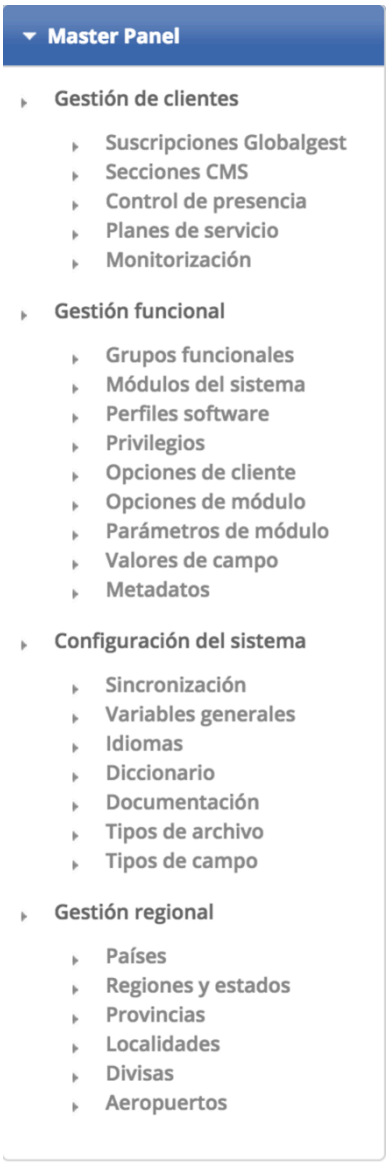


Figura 153. Master Panel en Globalgest

La Figura 153 es una captura de pantalla en la que podemos ver desplegado el Master Panel con sus diferentes componentes. En ella podemos identificar:

- ***Gestión de clientes***: equivale al componente *Gestor de Tenants y Gestor Multi-target*. Además de dar de alta a clientes, definimos la suscripción de cliente. Destacamos la funcionalidad *Monitorización* que fue programada como soporte a la investigación de este trabajo y permite ver estadísticas de uso del sistema

por parte de los usuarios finales y su relación con la industria a la que pertenecen.

- ***Gestión modular***: se corresponde con el gestor del portfolio funcional. A través él, el usuario UNA puede dar de alta nuevas funcionalidades, definir sus características y particularidades.

- ***Configuración***: Despliega funcionalidades relacionadas con la configuración del entorno administrativo MT^2 (idiomas, diccionario, tipos de archivo permitidos, etc.).Además de estas, destacamos la presencia del módulo de *Sincronización* que permite la exportación y creación de copias de seguridad de los datos.

- ***Gestión regional*:** Tablas maestras comunes a todos los tenants pertenecientes a la capa de procesamiento común (CBP). Ejemplos de estas tablas son países, días de la semana, meses, etc.

4.1 Escalabilidad funcional

El nivel administrativo también es escalable para agregar nuevas funcionalidades con el tiempo. A medida que se detectan nuevas necesidades susceptibles der modeladas como CBP, son incorporadas.

Por ejemplo, la última funcionalidad CBP añadida como módulo ha sido la gestión de *Aeropuertos*. Tiene como finalidad llevar un registro de aeropuertos del mundo manteniendo sus datos que se almacenan en las bbdd administrativa: nombre, ciudad, país (clave foránea a la tabla CBP de *Países*), el código internacional y la zona horaria. Actualmente, la funcionalidad CBP almacena un total de 8107 aeropuertos de todo el mundo. La necesidad se detectó al solicitarse comercialmente el desarrollo de un módulo que controlara los *Vuelos* de sus trabajadores. Inicialmente hay una labor costosa de búsqueda, tratamiento y almacenamiento de información; sin embargo una vez terminado este trabajo no sólo mejora la experiencia de los usuarios de nivel de instancia, sino que unifica la información y además la tabla CBP puede ser aprovechada por otros módulos futuros.

Figura 154. Funcionalidad CBP en Globalgest para el control de aeropuertos

La Figura 154 es una captura de pantalla del módulo administrativo para la gestión de aeropuertos. Los administradores de Globalgest pueden añadir, editar o eliminar aeropuertos gestionando una tabla CPB que posteriormente podrá ser usada en módulos del nivel de instancia. En la Figura 155 vemos efectivamente el (re)uso de este componente CPB; el usuario final de un tenant que despliega la funcionalidad de Vuelos selecciona el aeropuerto de salida del trabajador.

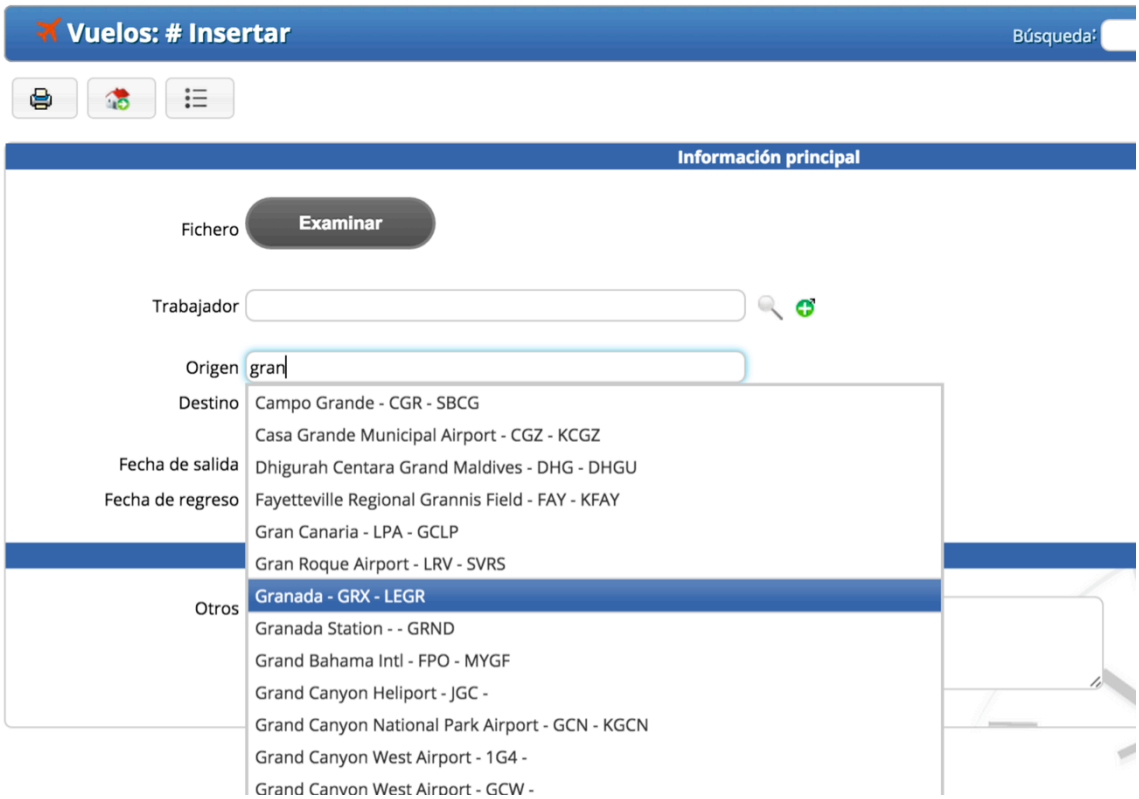


Figura 155. Módulo de vuelos de Globalgest desplegado en el nivel de instancia.

4.2 Gestión funcional

En Globalgest las funcionalidades están formadas por módulos de baja granularidad; de este modo, los tenants pueden seleccionar de forma más específica el subconjunto de ellos, para que su despliegue a nivel de instancia configure el SIE que más se adapte a sus necesidades.

Los módulos se encuadran dentro de un determinado *perfil* de sistema de información (CMS, CRM, ERP, etc.) y de un *grupo funcional* (ventas, compras, marketing, etc.). Ambas características son buenos indicativos del tipo de SIE que despliegan, además de servir como dimensión en la inteligencia de negocio para el vendedor (por ejemplo, si los tenants demandan más CMS que CRM, módulos de ventas o de compras, etc.). Incluso, son de gran interés comercial ya que cuando se desarrolla un módulo nuevo, se puede cotejar su perfil y compatibilidad con el de los ya contratados para ofrecerlo así a suscriptores específicos y aumentar las oportunidades de venta.

Como ejemplos de módulos tenemos *Clientes*, *Facturación* o *Presupuestos,* que en el caso concreto de Globalgest pertenecen al perfil "ERP" dentro del grupo funcional "Ventas". Estos módulos son horizontales e independientes del sector, pero en Globalgest también podemos encontrar módulos verticales dependientes de industria, como es el caso de las *Certificaciones de obra* (para el sector de la construcción) o *Citas de pacientes* (sector sanitario). La capa de presentación de Globalgest hace uso del atributo grupo funcional de los módulos para agruparlos en el menú de la aplicación.

Actualmente Globalgest permite desplegar 238 módulos agrupados en 42 grupos funcionales pertenecientes a 13 perfiles software

En el caso particular de Globalgest, el atributo perfil también contiene la industria a la que pertenece; valores ejemplo son: "ERP Constructoras", "ERP sanitario", "CMS Genérico", etc. No se lleva una relación módulo-industria sino que desglosan los perfiles software por industria. La gestión de tenants si que almacena la industria del cliente que a su vez se tiene en cuenta para el análisis de ventas.

Tanto los perfiles, industrias como grupos funcionales son totalmente configurables a su vez por módulos. Esto es, el administrador del sistema puede editar, eliminar o añadir nuevos valores. No sólo eso, tal y como explicaremos más adelante el control de

suscripciones también está articulado como un módulo de sistema. De este modo se podrá plantear en un futuro la existencia de *resellers*.

La Figura 156 representa una captura parcial de una instancia de aplicación; en ella se pueden apreciar 4 grupos funcionales: ventas, compras, productos e inteligencia de negocio. En la captura se detalla el grupo de ventas y en él podemos ver cómo el cliente tiene contratados 11 módulos.

Figura 156. Grupos funcionales y módulos en Globalgest desplegados en el nivel de instancia

4.3 Gestión de suscripciones

Las suscripciones son controladas por el módulo Gestor de cuentas de Cliente dentro del Master Panel. En Globalgest, existe una tabla que representa los metadatos MT^2 relacionando los módulos funcionales con los tenants y las particularidades de dicha suscripción.

En Globalgest los módulos pueden tener **dependencias**, por ejemplo *facturación* necesita a *gestión de clientes* ya que son las entidades a las cuales se les factura; del mismo modo, el *control de admisiones* requiere a los *pacientes y doctores*. Por ello, en *control de módulos* dentro de la gestión funcional administrativa, debemos establecer estas dependencias para que la suscripción de clientes las tenga en cuenta y las incorpore directamente en los metadatos MT^2.

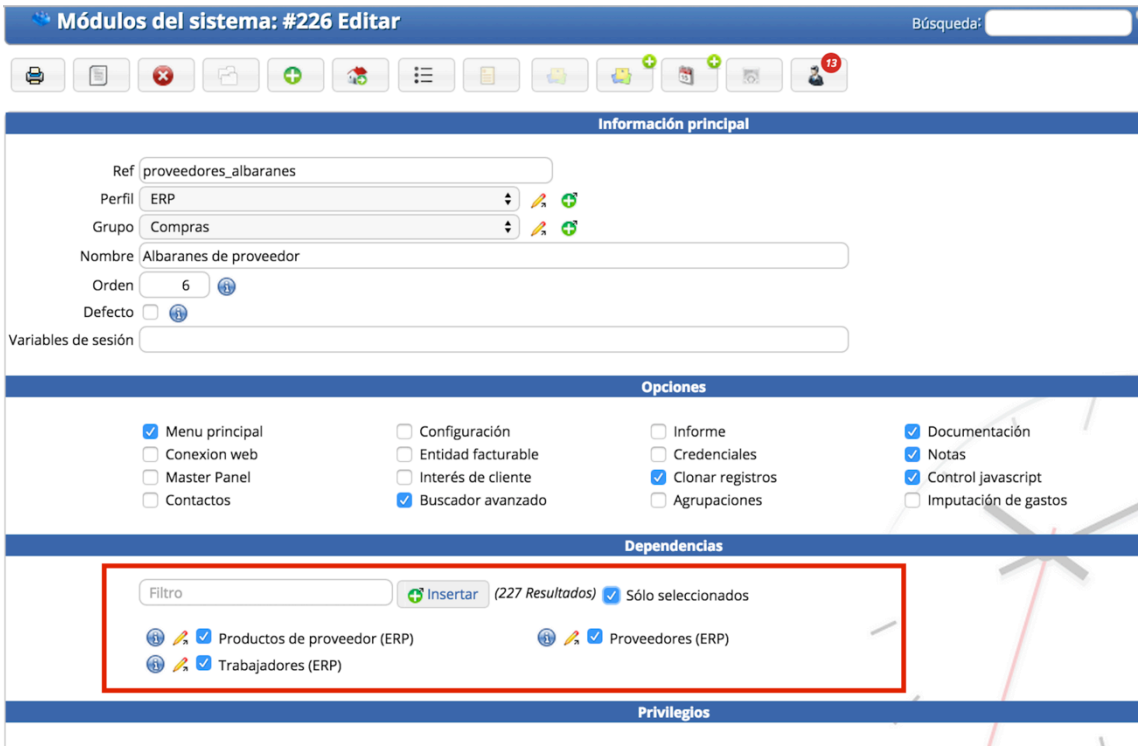


Figura 157. Definición de dependencias del módulo para Albaranes de proveedor

La Figura 157 nos muestra marcados en rojo el registro dentro del nivel administrativo para el control del módulo *albaranes de proveedor*. La finalidad de este módulo es almacenar los albaranes entregados por los proveedores tras la entrega del material. Lógicamente dicho módulo requiere la existencia de otros en la suscripción de cliente para su correcto:

- *Proveedor* del material.
- *Productos del proveedor* que se agregarán a las líneas de albarán.
- *Trabajador* que ha ocasionado la compra para el cálculo de rendimientos

En la Figura 158 tenemos el módulo de albaranes de proveedor en ejecución para un determinado tenant. En la captura podemos observar en los recuadros en rojo las relaciones con los módulos dependientes. La contratación del módulo de albaranes de proveedor no tiene sentido si no existen en el contrato los módulos anteriores. Es por ello por lo que cuando el vendedor de Globalgest configura una nueva suscripción y marca el módulo albaranes de proveedor, automáticamente se seleccionan los módulos dependientes (Figura 159).

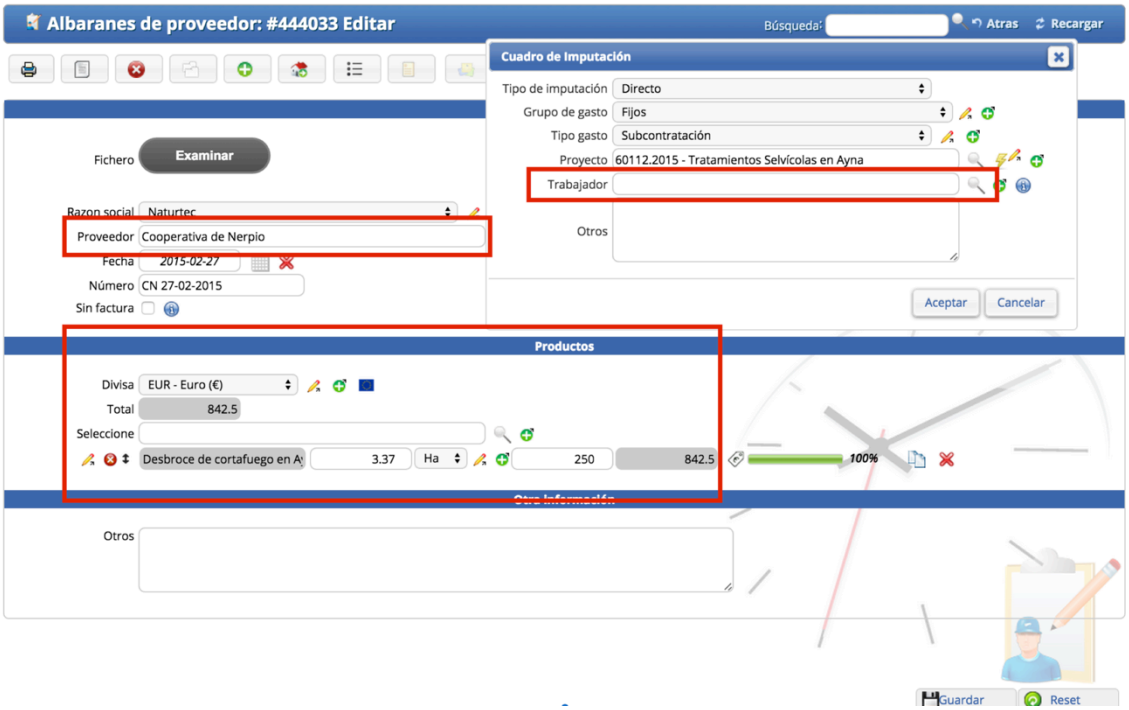


Figura 158. Despliegue del módulo albaranes de proveedor en el nivel de instancia.

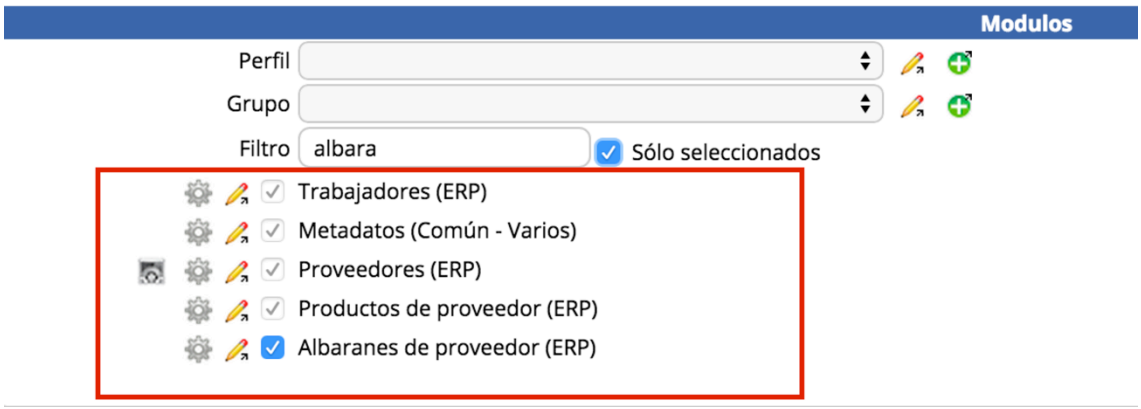


Figura 159. Las dependencias obligan a la contratación de módulos adicionales.

La Figura 160 es una representación gráfica de cómo los metadatos MT^2 almacenan las suscripciones de dos empresas de distinta índole (tecnología y medicina). Por cuestiones de espacio el diagrama sólo muestra los grupos funcionales. Como podemos apreciar en la parte izquierda, existen módulos relacionados con el control de pacientes que sólo son desplegados por la clínica médica. En el caso de la empresa de nuevas tecnologías, la suscripción es casi idéntica pero con pequeñas diferencias; obviamente esta empresa no necesita desplegar módulos médicos (pacientes, compañías aseguradoras, etc.), pero por el contrario si incluye en su portfolio módulos CMS para el control y gestión de la página web de la empresa. En la parte derecha de la figura se pueden apreciar ambos

niveles de instancia, cada uno con los módulos contratados y una capa de presentación personalizada.

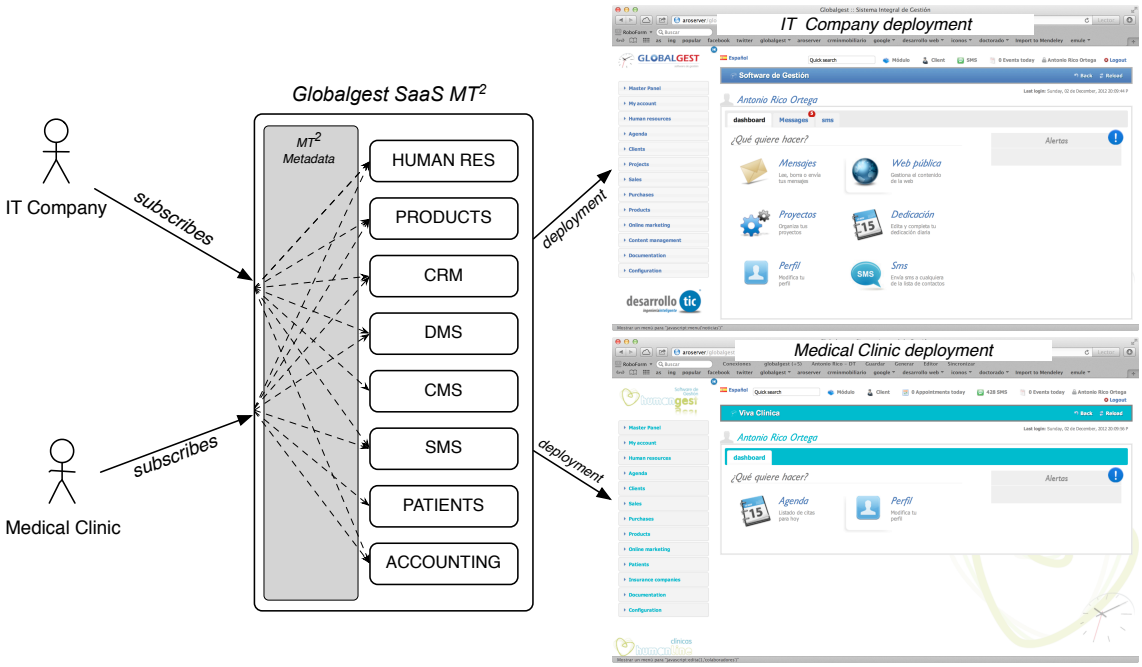


Figura 160. Suscripciones de empresa IT y médica y sus respectivos despliegues.

La suscripción a un módulo específico puede que requiera configuración extra; por ejemplo, la funcionalidad de *citas* fue desarrollada con la posibilidad de enviar mensajes SMS recordatorios a los pacientes. En este caso, al seleccionar la funcionalidad SMS cuando configuramos la suscripción, debemos establecer también el número de mensajes contratados.

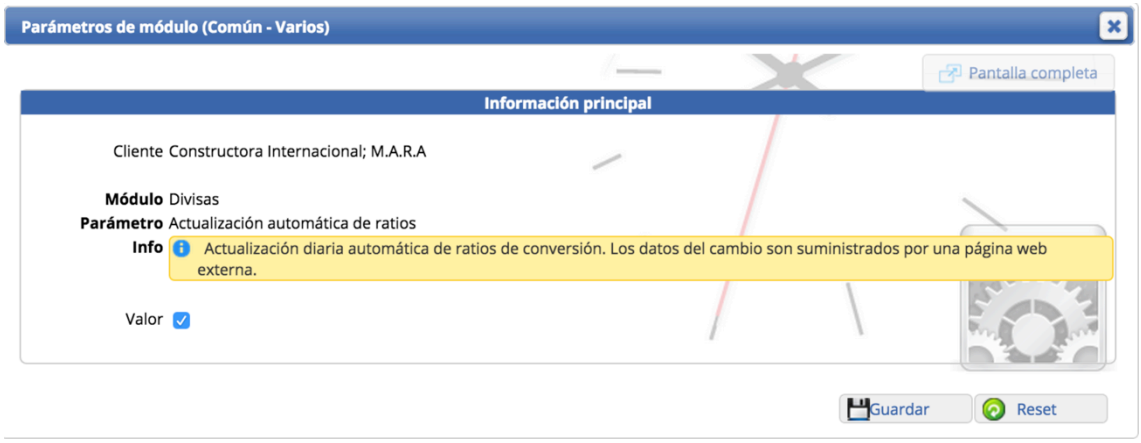


Figura 161. Configuración específica del contrato funcional del módulo divisas para un tenant.

Otro ejemplo es el módulo de *Control de divisas* cuya activación permite llevar un registro de divisas y sus ratios de conversión (ver Figura 161). Su activación habilita cambios en otros módulos (facturas, albaranes, etc.) y en el contrato funcional de cliente tenemos la posibilidad de indicar si queremos que el cambio de la divisa se calcule automáticamente (actualizándose mediante un servicio web) o queremos que sea el cliente el que lo establezca.

Posteriormente veremos en la sección de Agilidad en implantación un ejemplo más detallado de los pasos a seguir para dar de alta a un tenant en el sistema.

4.4 Un mercado multi-target

El portfolio funcional en Globalgest aumenta con las solicitudes de los clientes. A medida que los tenants piden nuevos desarrollos, estos son incorporados como módulos dentro del sistema y ofrecidos como servicio al resto de tenants. De este modo, el tenant original que pidió la funcionalidad gasta menos dinero (es más económico ya que se puede revender a otros clientes) y el propietario de Globalgest mejora la calidad del software (en algunos casos incluso abre nuevos mercados al una funcionalidad de una industria nueva). Por ejemplo, el módulo de *admisión de pacientes* fue inicialmente programado para una clínica médica en Sotogrande; ahora el sector sanitario está al alcance de Globalgest y un tenant nuevo (una clínica dental de Granada) se está beneficiando de la funcionalidad. Incluso, aún si la funcionalidad demandada es tan específica y a medida que no se pueda reutilizar por otros clientes, siempre puede ser vendida a precios normales y aprovechar la agilidad en el desarrollo que proporciona la capa CBP.

Tal como puede verse en la Figura 162-a, cuando un cliente potencial está interesado en el sistema, el proveedor de Globalgest examina si las funcionalidades que necesita la nueva empresa pueden ser articuladas con los módulos del portfolio. Si no es posible, entonces las funcionalidades son desarrolladas e incorporadas al sistema aprovechando los componentes CBP de la capa de procesamiento común. El desarrollo es más sencillo que en otros sistemas, ya que básicamente se reduce al esfuerzo de programación de componentes IBP, propios de las nuevas funcionalidades. Cuando todo está listo, se establece el contrato de cliente especificando los módulos y términos de despliegue (se

configuran los metadatos MT^2). Una vez almacenada la suscripción funcional, se crea el usuario administrador del tenant y la aplicación está lista para ser usada.

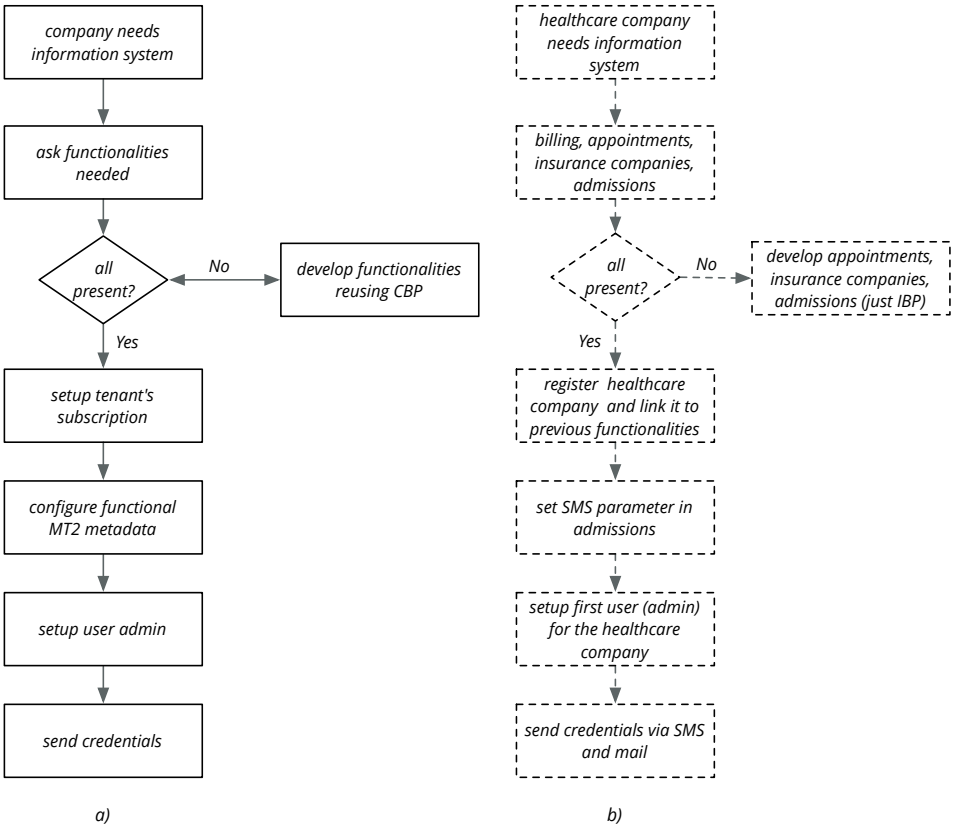


Figura 162. a) Pasos para alojar un nuevo tenant b) Caso real.

La parte derecha de la Figura 162 representa un escenario particular basado en un caso real. Una clínica médica necesitaba un sistema de información que cubriera las siguientes características: facturación, gestión de citas, admisionado de pacientes y compañías de seguros. El portfolio de Globalgest sólo cubría por completo la primera funcionalidad así que el resto era necesario desarrollarlas. Sin embargo, dado que existía un módulo de *gestión de eventos calendario*, el módulo de control de citas y admisionado no era necesario hacerlo desde cero. En el CBP del sistema existía librerías que no necesitaban ser programadas. Además, la gestión de compañías aseguradoras resultó ser muy similar a la de proveedores, ya existente en el portfolio. Por tanto, muchos componentes CBP fueron compartidos y reutilizados y sólo los elementos IBP hubo que programarlos. Una vez que los módulos fueron terminados e incorporados al sistema mediante el gestor funcional, la clínica fue registrada en el sistema y configurado su contrato funcional añadiendo estas nuevas funcionalidades. A partir de

este momento, nuevos módulos pertenecientes al sector sanitario podían ser ofrecidos a otros clientes (existentes o potenciales).

De este modo, el desarrollo de nuevos módulos permite a Globalgest escalar no sólo a nivel funcional sino que también a nivel de industrial ampliando el mercado de clientes potenciales. En la actualidad, una única instancia de aplicación es compartida por empresas pertenecientes a sectores dispares como una clínica médica, una constructora o una empresa IT de servicios informáticos. La Figura 163 representa un subconjunto del portfolio funcional presente en Globalgest y los recuadros representan los módulos funcionales. Los módulos de color blanco están presentes en los contratos de todas las empresas, los azules se ejecutan en la instancia de la de la empresa IT, los naranjas son desplegados por la constructora mientras que los verdes pertenecen al contrato funcional de la clínica médica.

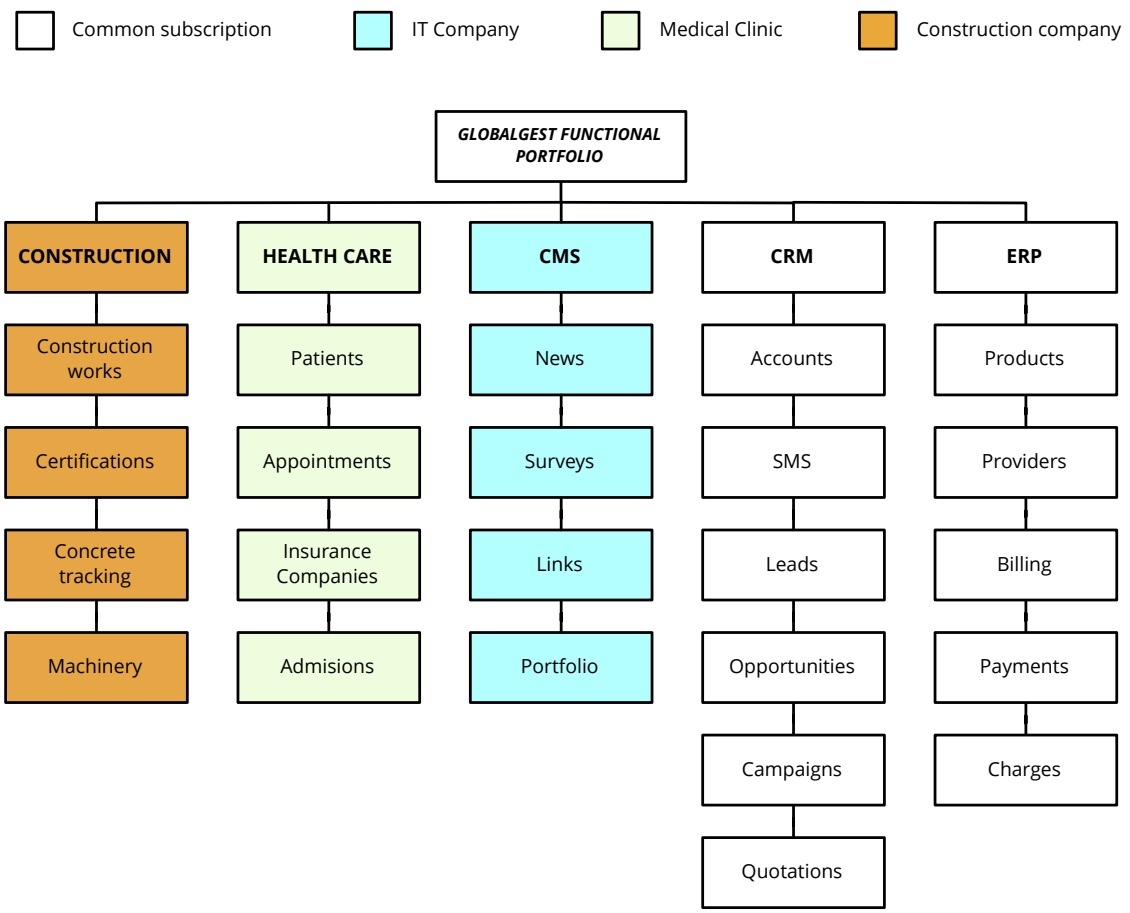


Figura 163. Globalgest ofrece servicios a múltiples sectores industriales

Tal y como vemos en el ejemplo, existen funcionalidades intersectoriales relacionadas con actividades comunes (facturación, proveedores, pagos, cobros, etc.) e idóneos para

ser contratadas por cualquier empresa mientras que hay otros módulos verticales, específicos de industria y altamente relacionados por ese sector.

La granularidad funcional a través de módulos permite a Globalgest suministrar servicios semejantes, en diferentes niveles de complejidad. Por ejemplo, la facturación de una empresa puede ser ampliada con el módulo de *control de cobros*. Este modulo mejora la funcionalidad permitiendo establecer cobros parciales sobre las facturas (con fecha, forma de pago, etc.) hasta que el importe total de la factura sea satisfecho. Puede que haya clientes a los cuales no les interese este control tan desarrollado, lo cual no quiere decir que no puedan llévalo a otro nivel. En caso de que la facturación no tenga el módulo de control de cobros activo, el sistema simplemente mostrará una casilla booleana para indicar si la factura ha sido abonada o no.

Por tanto, los sistemas MT^2 tienen un alcance mayor no sólo por la multitud de industrias a las cuales pueden dirigirse comercialmente, sino por el tamaño de las empresas. Empresas menores pueden usar módulos base sin ampliaciones modulares mientras que las necesidades de gestión superiores de mayores organizaciones pueden ser articuladas con el desarrollo de módulos de ampliación.

5 Nivel de instancia

Las aplicaciones que los tenants despliegan se corresponden con el nivel de instancia. En Globalgest, el modelo arquitectónico en este nivel es muy parecido al modelo genérico presentado en el capítulo anterior. En la Figura 164 lo podemos ver representado; en ella apreciamos cómo en este caso no se encuentran los metadatos de extensión, ya que el sistema todavía no permite que los tenants puedan personalizar la capa de datos. En la siguiente sección explicaremos brevemente el modelo de datos y el componente de transformación de consultas. No nos detendremos en más detalles dada su semejanza con los modelos de referencia expuestos en este trabajo.

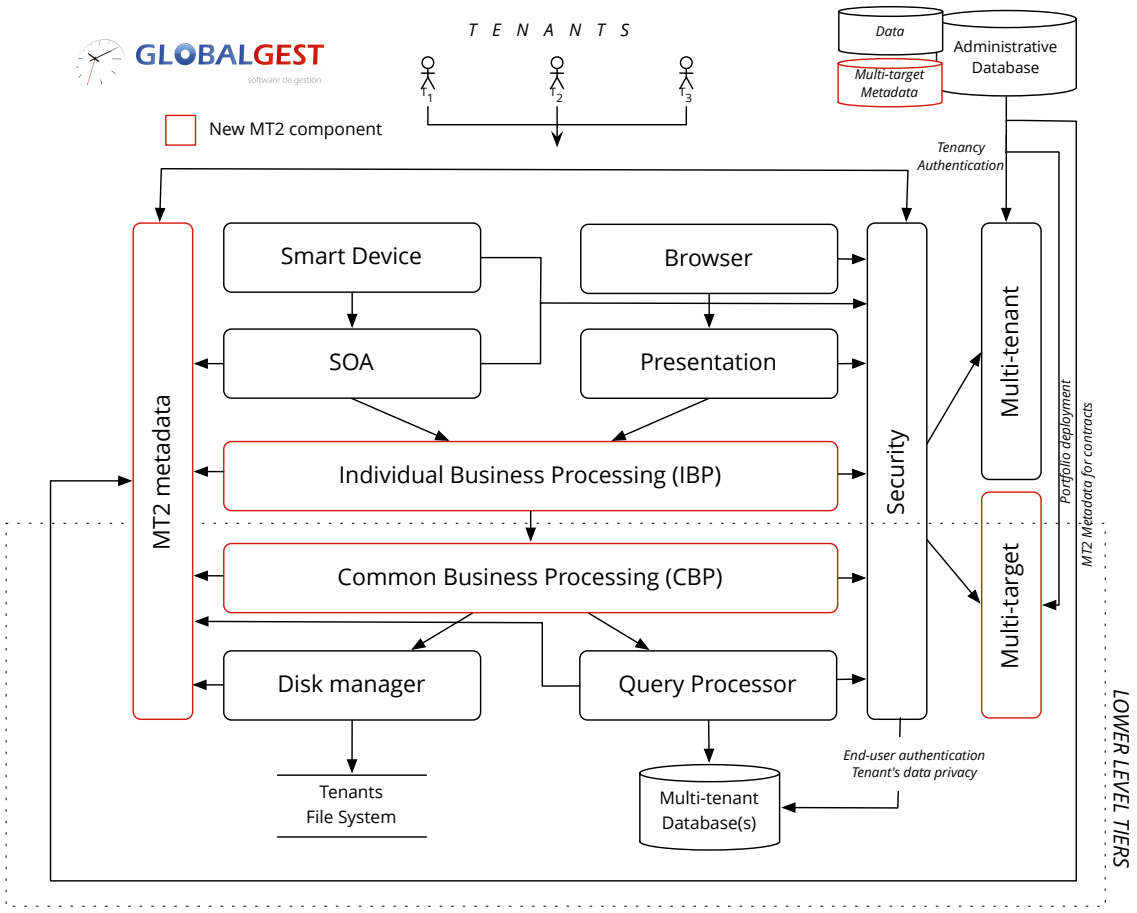


Figura 164. Globalgest: Arquitectura a nivel de instancia.

5.1 La capa de datos

Globalgest materializa multi-tenancy en la capa de datos con un enfoque compartido *Basic Layout* [57]. Siguiendo la nomenclatura ya utilizada en este trabajo, veamos un

ejemplo de *Tabla Tenant* (TT - Tenants Table) y la *Tabla Administrativa de Tenants* (ATT, Administrative Tenants Table).

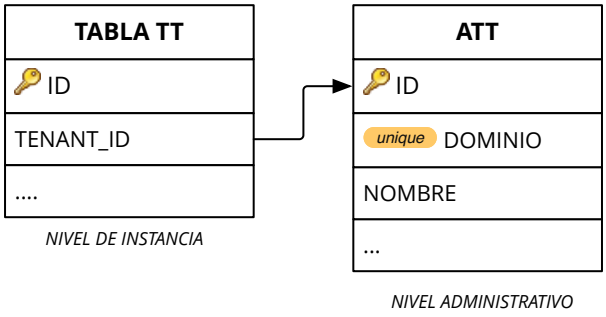


Figura 165. La Tabla Tipo en Globalgest arrastra en cada registro una clave foránea de arrendatario

Tal y como vemos en la Figura 165, los suscriptores comparten las tablas en la base de datos y su identificador de arrendatario es arrastrado en cada uno de los registros. Este campo (Tenant_ID) es la clave foránea a la ATT, la cual contiene el registro de todos las organizaciones suscritas a Globalgest. El campo *dominio* es un campo único y se corresponde con el campo *Tenant_CAD* que estudiábamos en el capítulo anterior.

En cada consulta a la base de datos el CTC usa el campo Tenant_ID para filtrar los registros que pertenezcan al arrendatario conectado. Veamos por ejemplo la función *getRows*. Esta función pertenece a la librería de funciones presente en la CBP; su cometido es devolver los registros de una determinada tabla dada una condición de filtrado.

```
function getRows(string $Tabla, string $Cond, integer $Tenant){
      // Condición final: Se añade el filtrado por Tenant
      $cond_ctc = $Cond.” and tenant_id=”.$Tenant;
      $sql = “select * from ”.$Tabla.” where “.$cond_ctc;
      return mysql_query($sql);
}
```

Tal y como podemos apreciar en la función *getRows*, el CPC siempre añade a las consultas una condición extra que asegura la privacidad de los datos entre tenants.

Las relaciones entre tablas se encuentran reflejadas en la Figura 167; además del identificador de tenant, las tablas de los clientes presentan cuatro campos comunes más:

- ***UIS***: Usuario de inserción en el sistema. Identifica al usuario final que insertó el registro, no se puede modificar.
- ***FIS***: Fecha de inserción en el sistema. *Time Stamp* que permanece invariable una vez relleno y que indica el momento en que UIS insertó el registro.
- ***UMS***: Usuario de modificación en el sistema. Enlaza con el usuario último que editó el registro. En cada modificación, el campo es actualizado.
- ***FMS***: Huella de tiempo del cambio realizado por UMS.

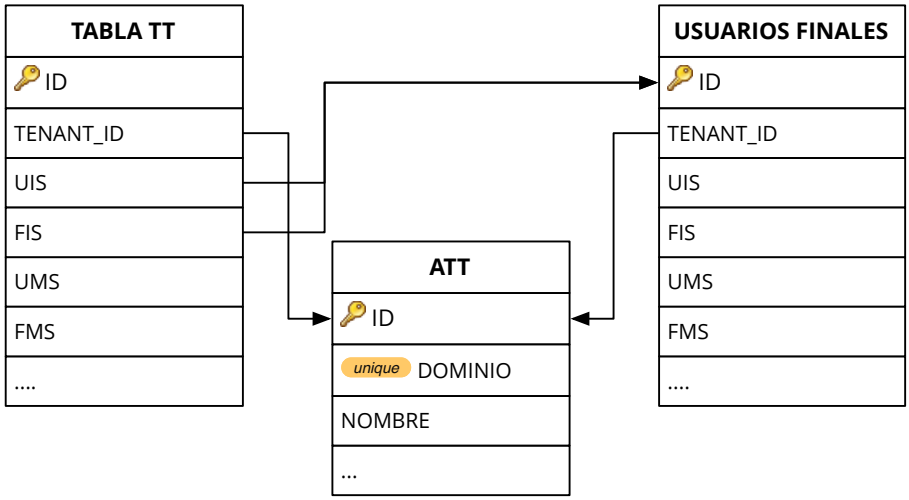


Figura 166. Relaciones de tablas TT-AT-USUARIOS en Globalgest

Gracias a estos nuevos campos y tal como muestra la Figura 167 podremos mostrar en la ficha de cada registro qué usuario lo insertó y quién fue el último que lo modificó:

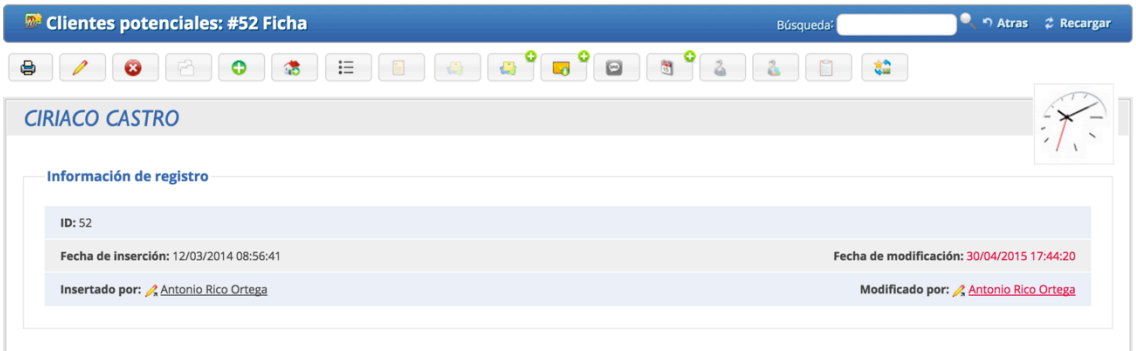


Figura 167. La información de registro muestra el usuario de inserción y de modificación

Cada una de las tablas tendrá, además de estos campos base, una serie de campos y relaciones personalizadas dependiendo de la funcionalidad que despliegan.

5.2 Personalización del sistema

Los suscriptores de Globalgest pueden personalizar la instancia de su aplicación a dos niveles: nivel funcional, y nivel de presentación; vemos cada uno de ellos en detalle.

5.2.1 Nivel funcional

Tal y como hemos visto en las secciones anteriores el contrato funcional de los clientes establece los módulos contratados y en consecuencia las funcionalidades desplegadas en la ejecución de la aplicación. A pesar de que los módulos son los mismos, estos permiten un grado de configurabilidad vía metadatos que articula una experiencia única de usuario en el nivel de instancia. Con este fin, el nivel administrativo despliega dos módulos que afectan a dicha configurabilidad: las opciones de cliente y los parámetros de módulo.

Las *opciones de cliente* se definen por suscriptor y afectan de forma general al despliegue de cualquier funcionalidad. Ejemplo de opción de cliente es la ejecución de tareas programadas (cron). Al seleccionar dicha opción, el tenant despliega esta funcionalidad cuyo objetivo es llamar todos los días a scripts de código que completan ciertas tareas. Dichas tareas ya si que están asociadas a los módulos contratados y ejecutan trabajos diarios relacionados con las mismas. En caso del módulo facturación por ejemplo, el demonio cron comprueba las facturas pendientes de pago y envía recordatorios a los clientes deudores transcurridos 30 días desde la fecha de emisión de la factura.

Los *parámetros* sí se definen a nivel de módulo y representan particularidades del contrato para esa funcionalidad concreta. Aparte de casos obvios como el número de SMS contratados o licencias de usuarios del sistema podemos encontrarnos con otros no tan evidentes como los días para alerta de impago mediante cron o el módulo donde aplicar las imputaciones de costes (puede ser en facturas o en los albaranes de proveedor). En la Figura 168 podemos ver la captura en la que el usuario UNA define el parámetro de modulo; dicho cliente tiene contratado la funcionalidad imputaciones (para el control de costes) y dentro de ella, el nivel administrativo ha establecido los *albaranes de proveedor* como módulo donde procesarlas y por ende llevar el control. Como resultado, el despliegue en instancia de la funcionalidad *albaranes* muestra el cuadro de imputación (Figura 169) en cada líneas del albarán. Este cuadro no aparecerá

si el cliente no tiene las imputaciones contratadas o si el parámetro especificara que el control de gastos se debe llevar en las facturas de proveedor (en cuyo caso aparecía en las líneas de factura).

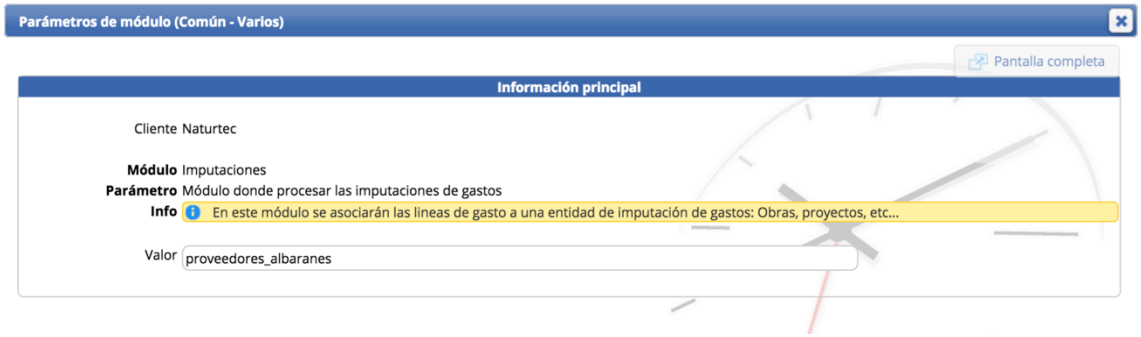


Figura 168. Instanciación de parámetro de módulo en el contrato de cliente

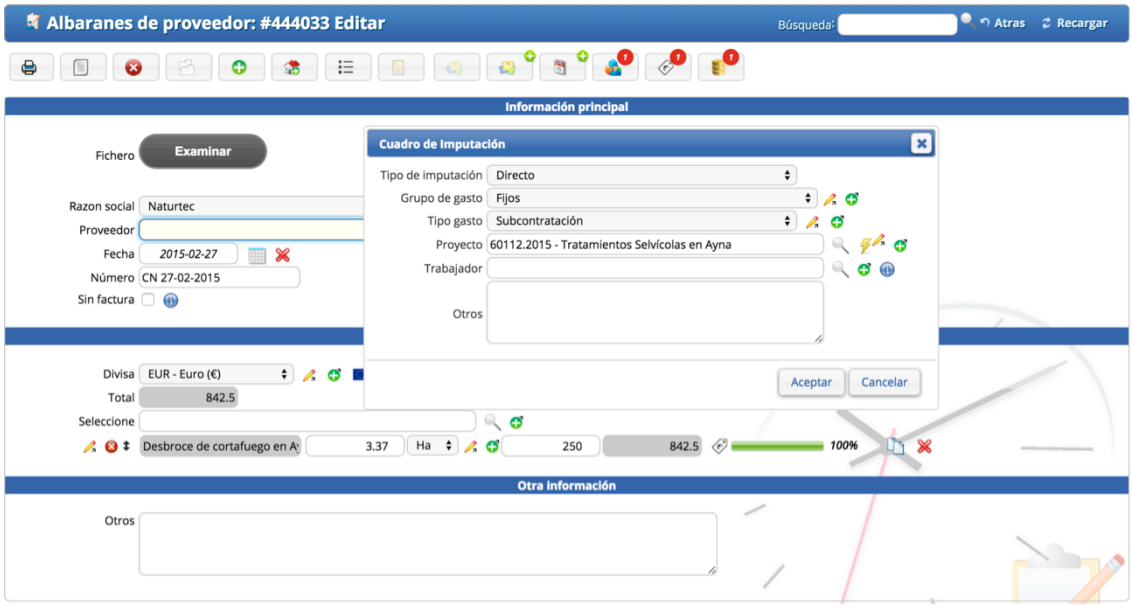


Figura 169. Despliegue de funcionalidad de albaranes de proveedor con imputación de costes.

5.2.2 Nivel de presentación

Globalgest permite adaptar el diseño y particularizarlo a la imagen corporativa del cliente suscriptor. Esto se consigue principalmente mediante el uso de hojas de estilo, y vía metadatos para personalización del menú de aplicación.

Las hojas de estilo CSS modifican la presentación de los contenidos en pantalla. Si el etiquetado HTML separa la información de la presentación, entonces podemos cambiar el diseño de la aplicación con tan sólo modificar la hoja de estilos por una adaptada al cliente. En Globalgest, el motor de ejecución permite incorporar automáticamente hojas

CSS específicas de cliente definidas en el nivel administrativo. Los CSS se incorporan en última instancia para poder sobrescribir las reglas establecidas por la hoja CSS genérica.

La Figura 170 muestra la personalización de interfaz realizada para la empresa sanitaria. En ella vemos como predominan los colores verdes y logotipos de la propia marca frente a la Figura 171, donde se ha cargado el CSS genérico de la aplicación. En ambas figuras también podemos ver cómo se presentan funcionalidades diferentes. Esto es debido a que el *dashboard* del inicio carga gráficas y *widgets* relacionados con los módulos contratados.

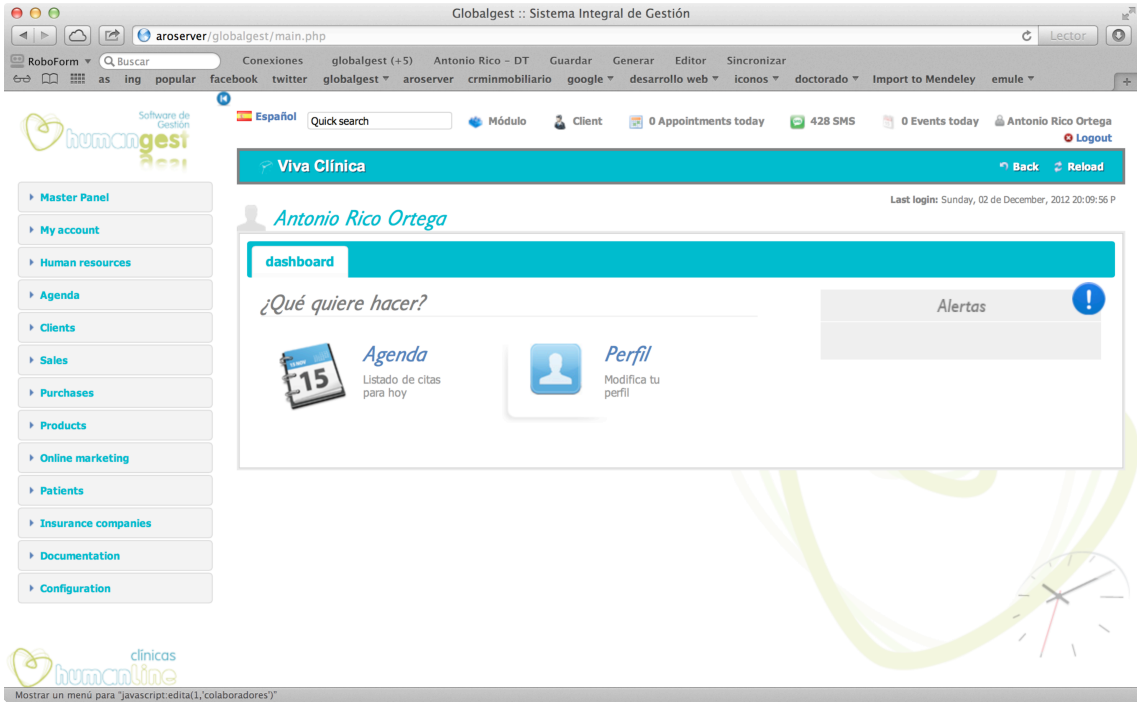


Figura 170. Nivel de instancia para clínica médica. Personalización de interfaz

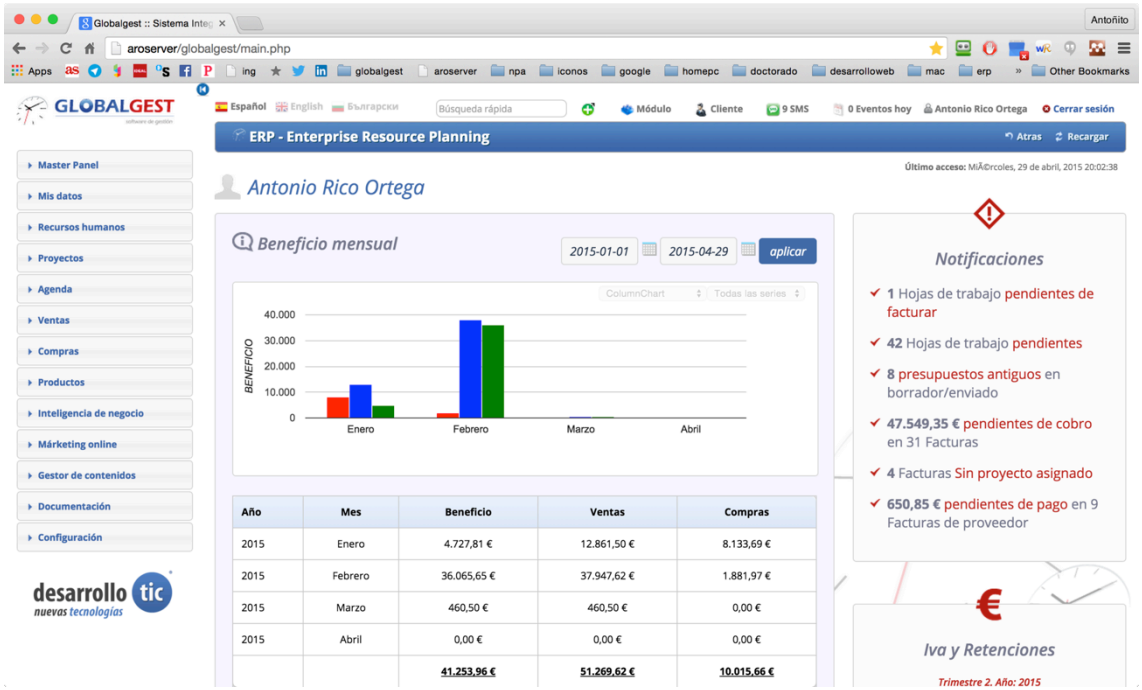


Figura 171. Nivel de instancia para empresa IT. Interfaz genérica

Los nombres de los módulos y el grupo funcional en donde irán encuadrados también pueden ser personalizados y alterados de su posición por defecto. Esta operación se realiza a nivel de módulo en el momento de la definición del contrato funcional y es almacenada en los metadatos MT^2. Así, el módulo proyectos puede pasar a llamarse "Obras" y aparecer encuadrado dentro de ventas cuando normalmente está en un grupo funcional del mismo nombre (Figura 172).

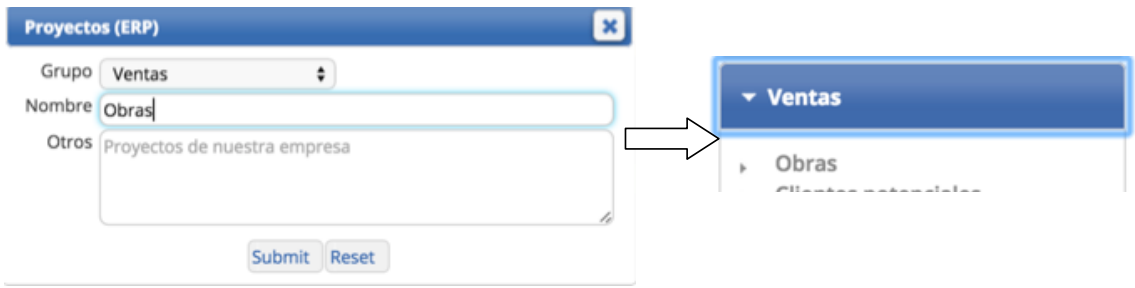


Figura 172. Los metadatos MT^2 son usados para la personalización del menú de la aplicación

Además del menú principal, Globalgest permite personalizar otros aspectos como el nombre del programa, la venta a de login, o los colores de texto y de fondo de los PDF.

Estas personalizaciones hacen que el cliente disfrute de una experiencia única de usuario, hacen transparente el entorno MT compartido y mejoran los argumentos de venta.

6 Agilidad

El entorno multi-funcional de Globalgest como software MT^2 incrementa de forma considerable el soporte a la agilidad. En efecto, la agilidad en la implantación que presentaba el modelo tradicional se ve favorecida debido a que funcionalidades distintas no implican diferentes implantaciones. Por otro lado, el patrón arquitectónico también apoya con fuerza al desarrollo ágil debido a la reusabilidad de componentes comunes. Veamos ejemplos concretos de estos beneficio explicando el alta de un cliente por un lado y por otro presentando los elementos CBP de la capa de procesamiento.

6.1 Agilidad en implantación

Para dar de alta un nuevo cliente basta con acceder como usuario UNA al sistema y clicar en Master Panel → Clientes → Insertar nuevo registro

Figura 173. Agilidad en implantación de Globalgest: alta de una nueva suscripción

En el formulario de inserción debemos rellenar un bloque de información de la empresa. El campo *dominio* es una clave única para cada arrendatario que nos permitirá definir una dirección URL de acceso al sistema. Destacar que se puede personalizar en este momento:

- Nombre del programa que posteriormente aparecerá en el nivel de instancia al arrendatario.
- Logotipo de la empresa.

- Color de fondo y de fuente en generación de documentos PDF como facturas, albaranes, etc.
- Divisa por defecto.
- Cuota de espacio en disco.
- Número de decimales en los redondeos.
- Número de usuarios/licencias.

Figura 174. Formulario de inserción de nuevo cliente en Globalgest

El usuario *admin* del tenant es el único usuario final que necesitamos dar de alta inicialmente (Figura 175), a partir de este y en el nivel de instancia se podrán crear el resto de usuarios hasta completar el límite establecido.

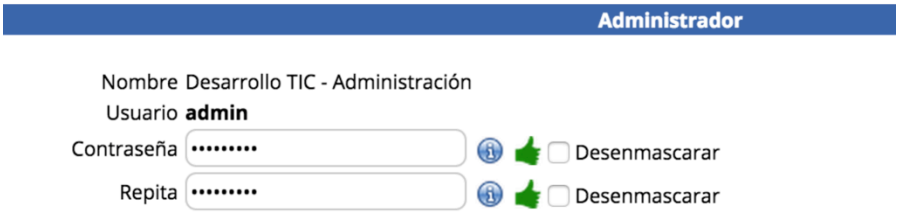


Figura 175. Alta de usuario admin

Hasta ahora los pasos seguidos pueden ser semejantes a los que tendríamos que realizar en el alta dentro de un software MT tradicional. Sin embargo la novedad viene del **establecimiento del contrato funcional de cliente** (metadatos MT^2). Para ello, tal y como vemos en la Figura 176 basta con marcar los módulos contratados

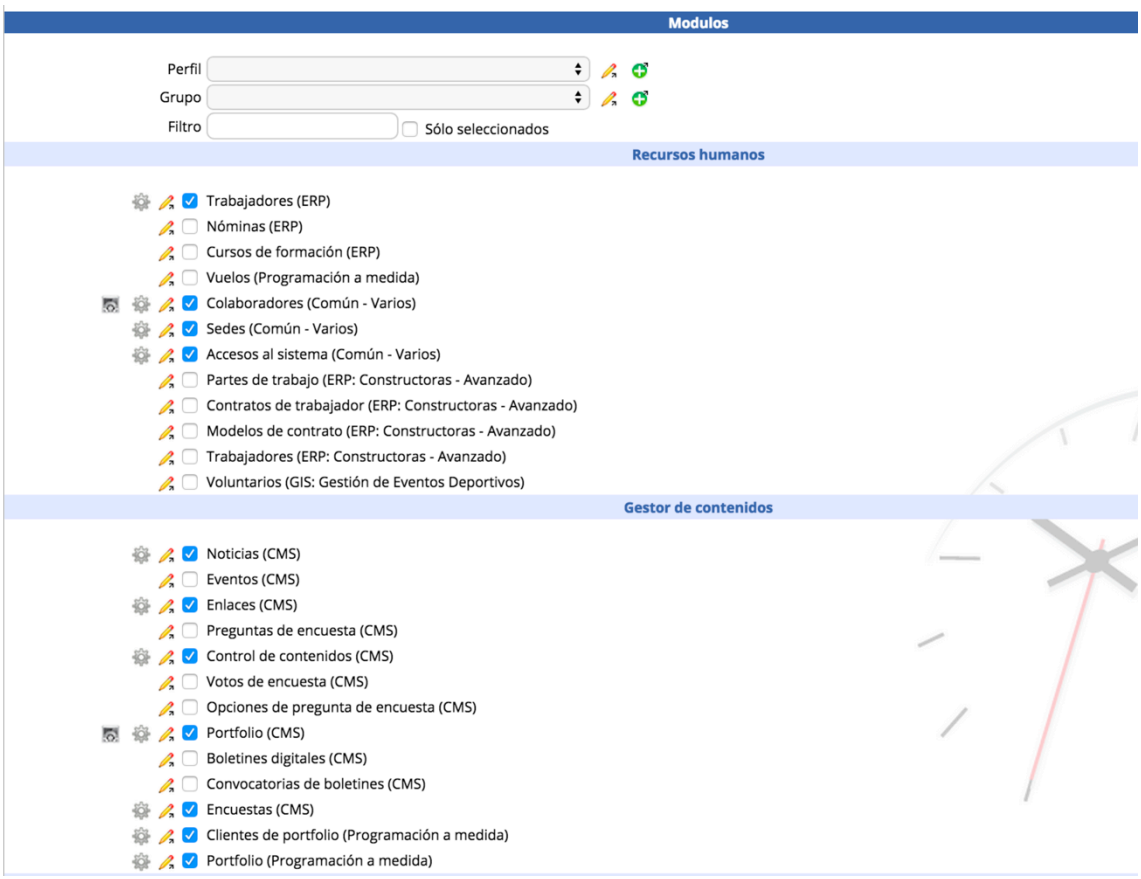


Figura 176. Globalgest: Selección de módulos para definir el contrato funcional del tenant

Existen funcionalidades que puede que necesiten parametrización. Para establecer valores a los parámetros en los metadatos MT^2 basta con clicar en el icono situado a la derecha del nombre del módulo. Este es el caso del envío de SMS (Figura 177), donde en la personalización del contrato debemos establecer el número de SMS comprados y la clave API de acceso a la pasarela de envío (Figura 178).

Figura 177. Personalización de metadatos MT^2

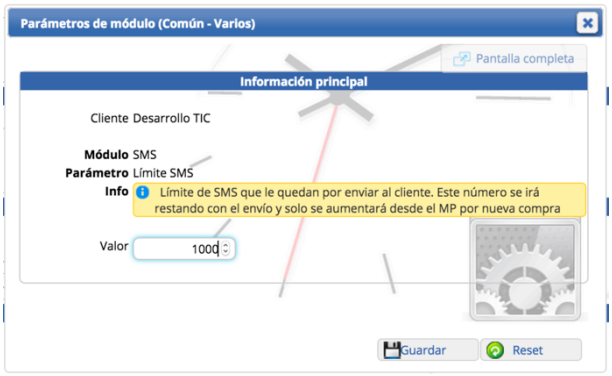
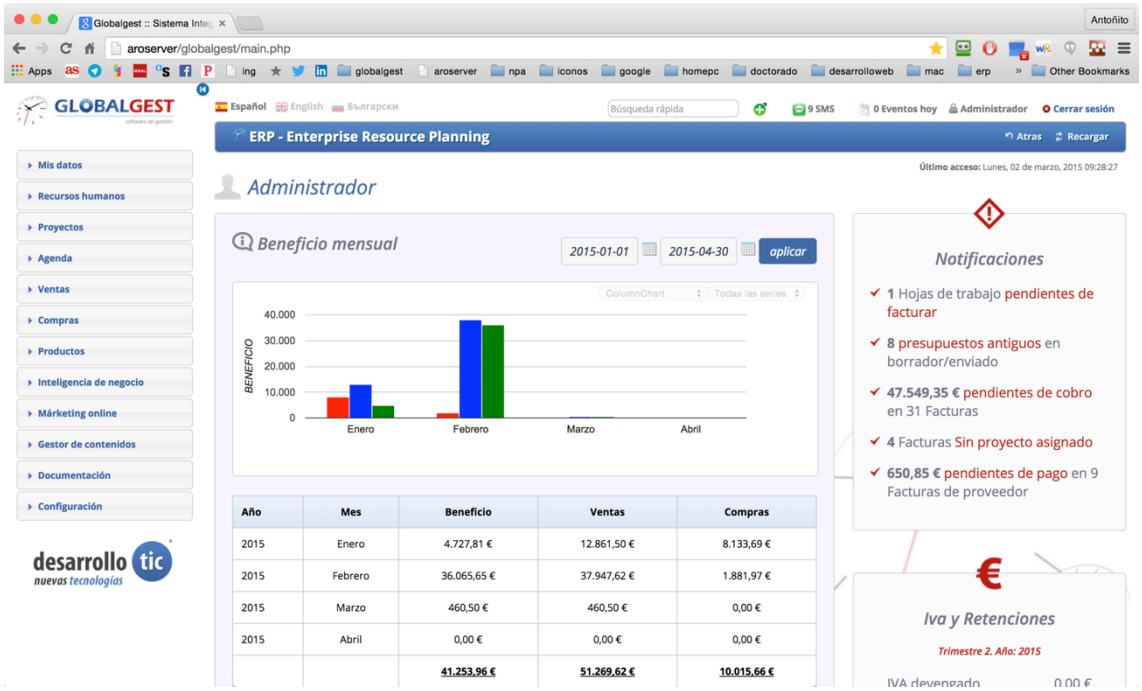


Figura 178. Personalización del módulo SMS mediante parametrización

En el mismo momento que los datos son guardados en el nivel administrativo, el usuario final *admin* del nuevo tenant ya podrá acceder al sistema; la operación de implantación de un SIE personalizado para la organización se ha realizado en cuestión de minutos (Figura 179).



Año	Mes	Beneficio	Ventas	Compras
2015	Enero	4.727,81 €	12.861,50 €	8.133,69 €
2015	Febrero	36.065,65 €	37.947,62 €	1.881,97 €
2015	Marzo	460,50 €	460,50 €	0,00 €
2015	Abril	0,00 €	0,00 €	0,00 €
		41.253,96 €	51.269,62 €	10.015,66 €

Figura 179. Instancia de Globalgest ejecutada por el usuario admin de la nueva cuenta

En Globalgest, la URL de acceso es única para cada suscriptor y se configura añadiendo el campo *dominio* de cada cliente a la URL de instalación. Este hecho permite también personalizar la capa de presentación para los clientes. Bastará con escoger una hoja de estilo con el mismo nombre que el dominio. Un ejemplo de pantalla de acceso personalizada lo podemos ver en la Figura 180.

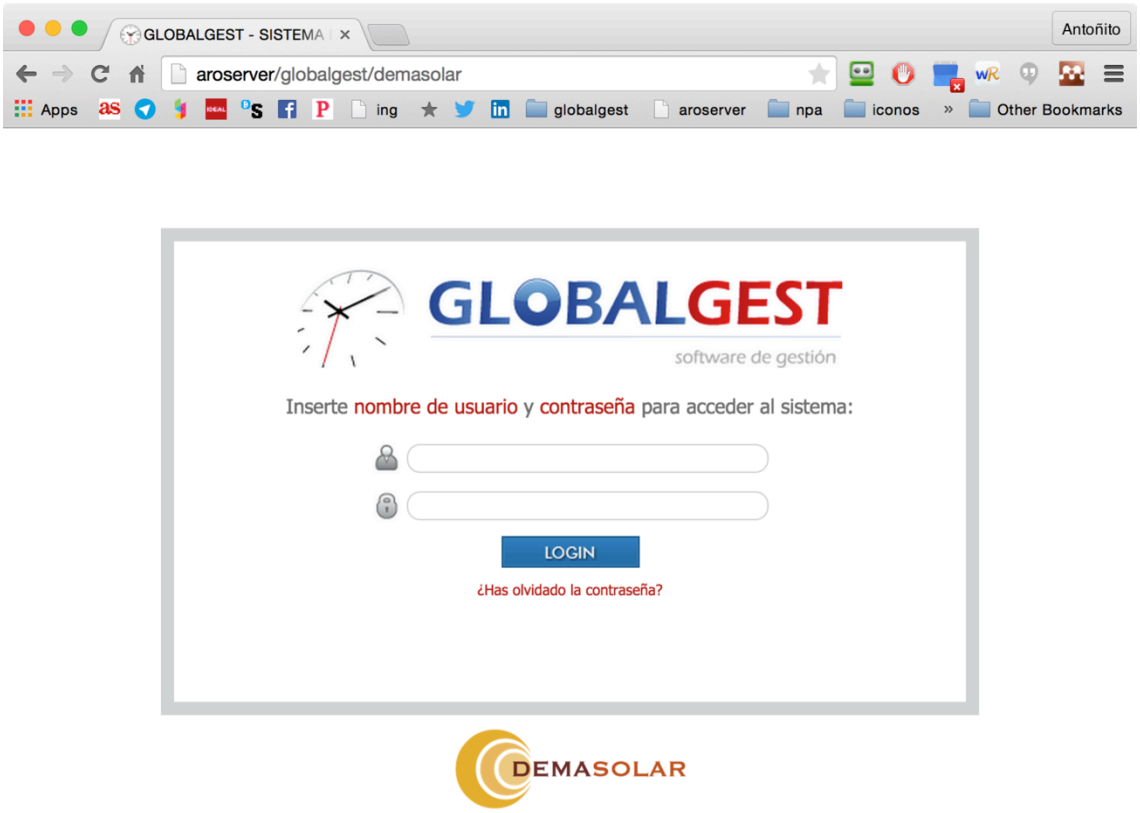


Figura 180. URL personalizada de acceso para el cliente Demasolar.

6.2 Agilidad en el desarrollo

Globalgest presenta una estructura de clases que puede ser reutilizada por los desarrolladores en nuevas funcionalidades. Se trata por tanto de componentes CBP que son importados estáticamente en tiempo de ejecución y están disponibles para ser reutilizados por cualquier despliegue.

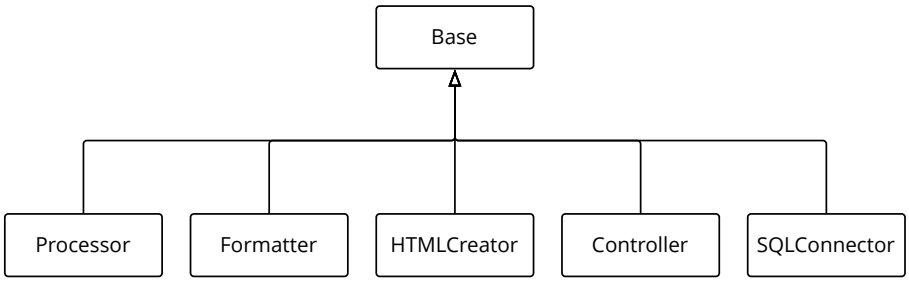


Figura 181. Diagrama de clases CBP en Globalgest

El diagrama de clases de la Figura 181 se corresponde con los elementos CBP que son importados de forma estática en todas las ejecuciones, independientemente del arrendatario y de las funcionalidades contratadas. Existe una clase *Base* que extienden

por herencia el resto de clases del sistema. Expliquemos detenidamente cada una de ellas:

- ***Base***: Funciones básicas del sistema. Desarrolla las funcionalidades principales de acceso a la base de datos (CTC) así como de depuración. El resto de clases hereda los métodos y atributos de la clase, pudiendo especializarlas y/o reutilizarlas.
- ***Processor***: Lógica de negocio común a todos los sectores.
- ***Formatter:*** Funciones de formateo de datos que sirve como apoyo al resto de clases del sistema. Entre ellas nos podemos encontrar: Formateo de fechas, exportación a otros formatos (Excel, RSS…).
- ***HTMLCreator:*** Métodos de presentación para las capa de acceso al sistema.
- ***Controller:*** Control de acceso a nivel de arrendatario(privacidad de datos y seguridad Multi-target), así como de usuario final.
- ***SQLConnector:*** Funciones complejas de acceso a la base de datos.

Una vez que un usuario se autentica correctamente en el sistema, Globalgest creará una instancia de cada una de ellas que permanecerá disponible en toda la ejecución.

```
<?
require("clases/Formatter.php");
require("clases/SQLConnector.php");
require("clases/HTMLCreator.php");
require("clases/Processor.php");
require("clases/Controller.php");

$f =  new Formatter($dominio);
$s =  new SQLConnector($dominio);
$h =  new HTMLCreator($dominio);
$p =  new Processor($dominio);
$c =  new Controller($dominio);
?>
```

Figura 182. Importación estática de componentes CBP en Globalgest

La Figura 182 es un ejemplo real del código de Globalgest. En él podemos observar cómo las clases CBP se instancian de forma estática, independientemente de las funcionalidades contratadas.

Teniendo como punto de partida estos elementos CBP reutilizables, la creación de una nueva funcionalidad (elementos de la capa IBP), no precisa formular de nuevo los métodos implementados; estos métodos son importados automáticamente en la

ejecución de instancia. Un desarrollador podrá extender las clases anteriores para adaptarlas a las necesidades de la nueva funcionalidad. Por **ejemplo**, la Figura 183 muestra una definición de clase IBP para la funcionalidad SMS que extiende *HTMLCreator* para mostrar un mensaje de error cuando el usuario destinatario no tiene teléfono asociado

```php
<?

class h_sms extends HTMLCreator{


    function sin_sms(){
        global $t;
        $html = $this->img("128.png",NULL,"",WWWBASE."modulos/sms/img/iconos/");
        $html.="<br/>".$t->get("error_no_telefonos_sms");
        return $this->popup($html);

    }

}


?>
```

Figura 183. Clase IBP h_sms que extiende a HTMLCreator

Este nuevo método *sin_sms* estará presente únicamente para los tenants con el módulo SMS contratado, ya que es importado dinámicamente en la ejecución de la funcionalidad. En la Figura 184 vemos el resultado en la ejecución de instancia.

Figura 184. Alerta de Globalgest si intentamos enviar un SMS a un contacto sin teléfono asociado

Al crear nuevas funcionalidades, todos los archivos deben ser almacenados en un mismo directorio. Este contendrá tanto los componentes de procesamiento individual (IBP), como de la capa de presentación específica de la funcionalidad (Figura 185).

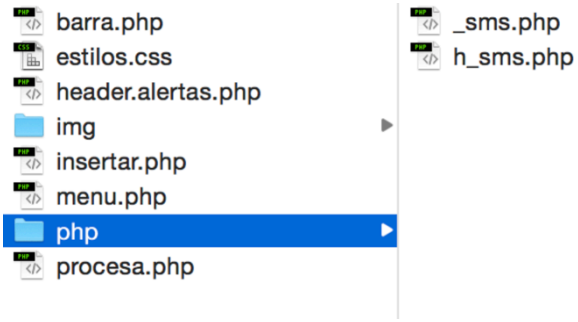


Figura 185. Archivos de la funcionalidad SMS

Finalizada la programación, podremos añadirla usando el *Gestor Funcional* de la capa de administración tal y como muestra la Figura 186. Una vez insertada, la funcionalidad ya estará disponible para el resto de arrendatarios presentes y futuros del sistema.

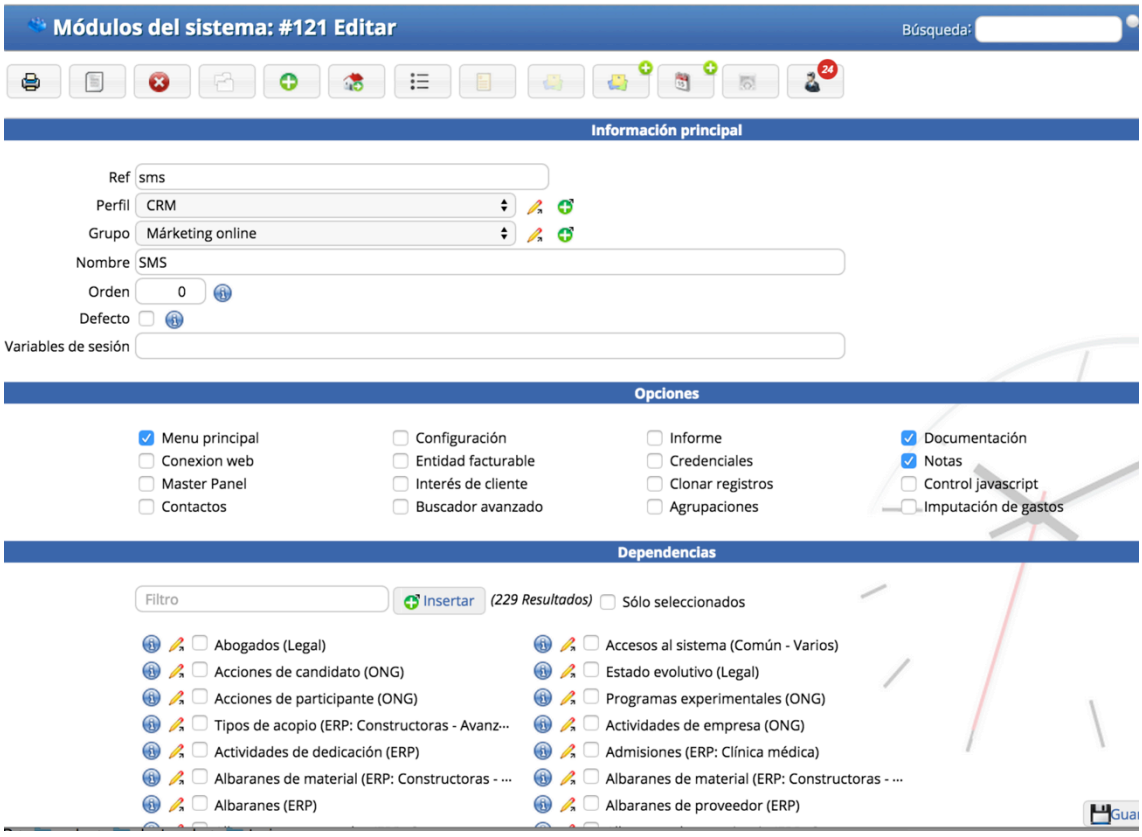


Figura 186. Alta de nueva funcionalidad en el Gestor Funcional

7 Globalgest y la PYME

Globalgest pertenece a una micro empresa de menos de diez trabajadores y aún así, es capaz de dar servicio satisfactoriamente a cerca de 40 empresas. Así, este software demuestra como MT^2 ayuda a las pequeñas empresas desarrolladoras a considerar SaaS como una fórmula de distribución de software. Recordemos que uno de los escollos en MT tradicional, era la dificultad que suponía el mantenimiento de un software distinto para cada funcionalidad ofrecida.

Con el objetivo de probar que este nuevo modelo permite a las empresas dar servicio a organizaciones dispares, llevamos a cabo un **experimento**. Dicho experimento fue realizado en el año 2013 durante 4 meses (desde Mayo hasta Agosto) y consistía en monitorizar el uso de los módulo por parte de los tenants suscriptores.

El experimento suponía el registro y monitorización de la actividad de los usuarios finales en el sistema. Durante este periodo, cada vez que la aplicación cargaba una página (Globalgest se accede vía web), los servicios de monitorización de la capa administrativa registraban información relativa al acceso: tenant, módulo, grupo funcional, perfil, industria y localidad geográfica.

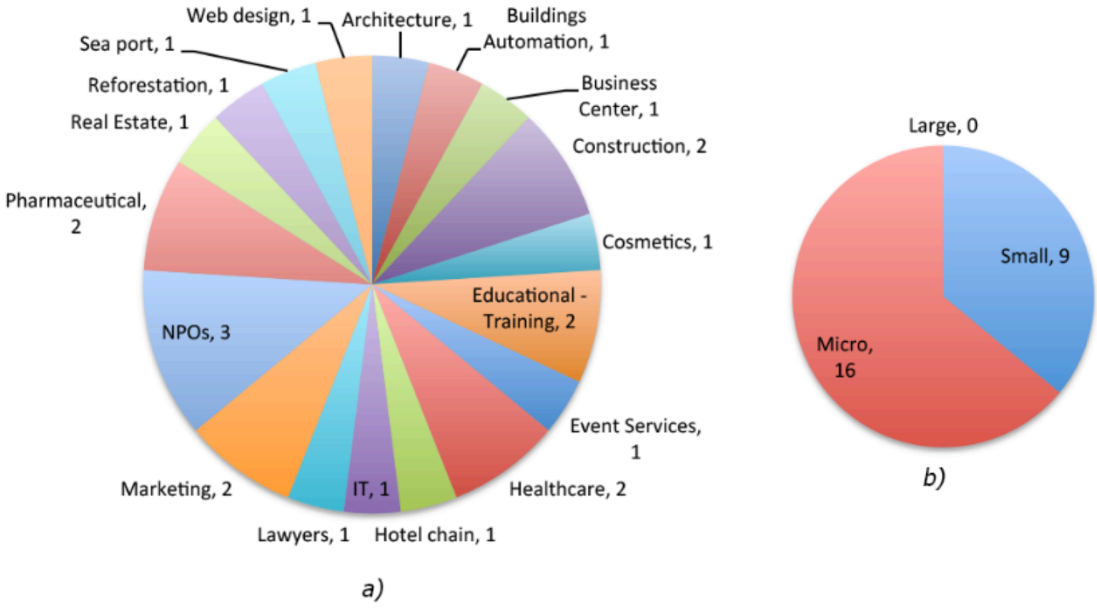


Figura 187. Distribución de clientes por sector (a) y por tamaño de empresa (b)

En total se registraron 58.638 accesos de usuarios finales de todos los tenants. Tal y como muestra la Figura 187, con un mercado multi-target Globalgest servía en aquel entonces ya a 25 empresas pertenecientes a 18 sectores de industria distintos. Entre los suscriptores no existen grandes empresas; nueve de ellos son micro-empresas con menos de 10 trabajadores mientras que el resto de tenants tienen un número mayor de empleados, pero siempre dentro de la categoría pyme. La mayor parte de las suscripciones tienen licencia para menos de 5 usuarios excepto una empresa constructora (que tiene 21 usuarios registrados) y una empresa de domótica con 11.

La Figura 188-a muestra la división por perfil de software que a su vez también identifica el sector industrial. Más de la mitad de los accesos pertenecen al mundo de la construcción, lo cual es consistente con el hecho de que sean las empresas con el mayor número de licencias. El perfil ERP genérico (conteniendo módulos multisectoriales como el de facturación) cubre el 25% de la muestra mientras que el resto de perfiles tienen un porcentaje análogo. Esto es lógico ya que las empresas, independientemente del sector siempre comparten actividades comunes. Esto es, facturación por ejemplo siempre se lleva a cabo, tanto si se una clínicas médicas como constructoras o estudios de arquitectura.

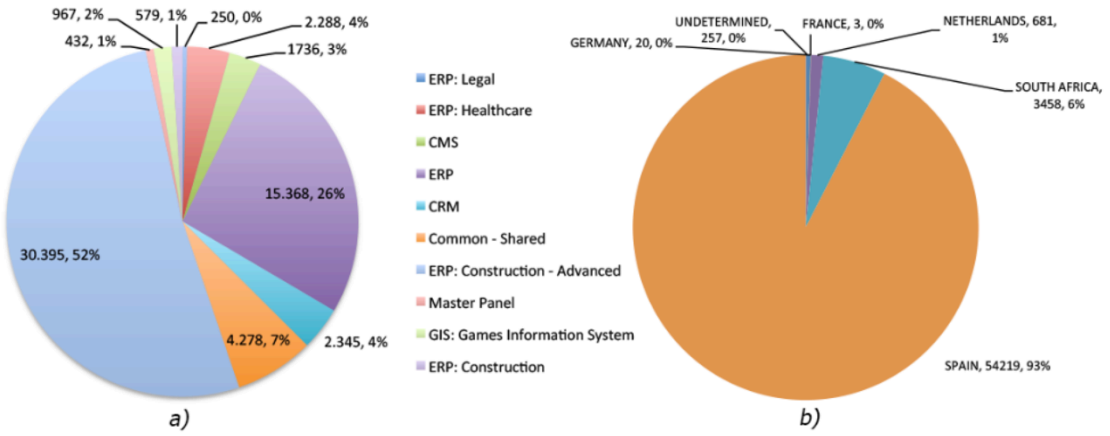


Figura 188. Monitorización de acceso en Globalgest: Perfil de Software (a) y Localización (b)

Al realizar este experimento, descubrimos una nueva ventaja que MT^2 aporta a las pyme como consecuencia de una migración a SaaS: la pequeña empresa ya no incluye el software como escollo en procesos de internacionalización. Gracias a la ubicuidad de la nube, las pymes pueden expandir su actividad a otros países pero mantener un sistema

de información común y centralizado. Esta oportunidad es clave especialmente en tiempos de crisis en los que las compañías siguen teniendo mano de obra y conocimiento pero sin embargo carecen de oportunidades de negocio y nuevos proyectos en su país.

Atendiendo a la Figura 188-b, vemos como existe un 6% de registros procedentes de Sudáfrica. Esto se debe a que uno de los tenants de Globalgest se expandió internacionalmente y existen usuarios finales en dicha localización. La mayor parte del resto de los accesos están focalizados en España debido a que el software Globalgest ha sido desarrollado por una empresa española y por tanto el foco comercial se ubica dentro de este país. El resto de la muestra refuerza la ventaja de ubicuidad en el uso de Globalgest con arquitectura cloud.

8 Ejemplos funcionales

Con menos de 5 años en producción, Globalgest es un sistema que actualmente está implantado en 37 empresas. En las secciones siguientes explicaremos algunas de las funcionalidades modeladas en el entorno MT^2 y que están siendo utilizadas en la actualidad.

Los siguientes ejemplos suponen de nuevo una herramienta de soporte y mantenimiento para este tipo de sistemas. Además, ilustran cómo Globalgest reutiliza diferentes módulos a través de distintas funcionalidades. Esta es la principal causa del uso de una granularidad baja, a fin de compartir módulos entre funcionalidades de distinta índole.

8.1 Control de dedicación

Figura 189. Frontend de control de dedicación

El control de dedicación de permite llevar un registro de la actividad de los trabajadores. Cada trabajador, debe imputar un número de horas diarias a determinadas tareas usando un frontend adaptado para móviles (Figura 189). De este modo, se podrán obtener informes de actividad y cálculo de rendimientos que nos ayudarán en la toma de decisiones a nivel corporativo. El control de dedicación ha sido diseñado según el diagrama de la Figura 190.

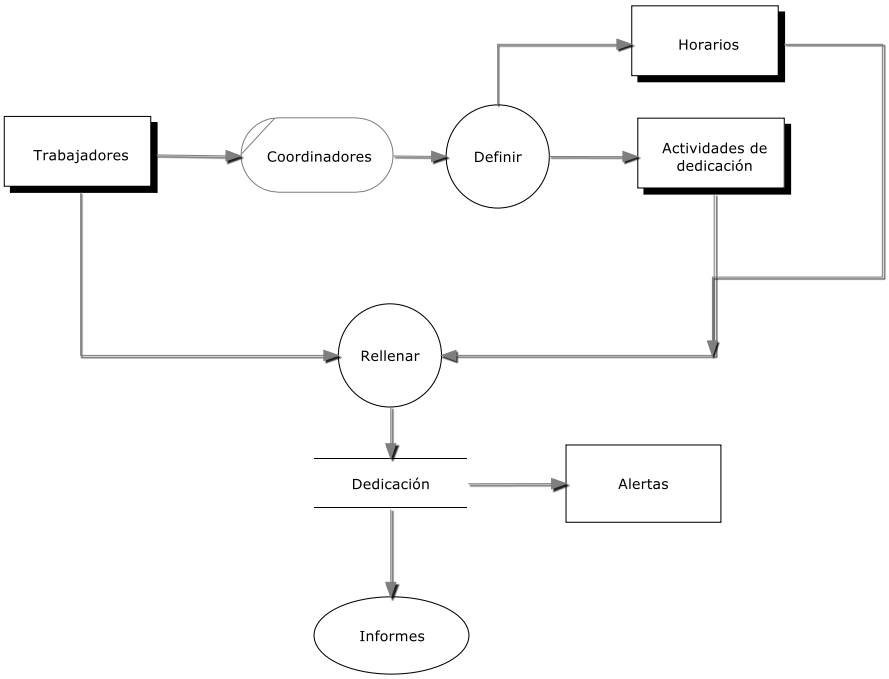


Figura 190. Diagrama para el Control de Dedicación en Globalgest

Los *trabajadores* deben rellenar diariamente una *dedicación* imputando una cantidad de horas de trabajo a cada *actividad de dedicación*. Las *actividades de dedicación* son definidas por un subconjunto de trabajadores con permisos llamados *coordinadores*. Los *coordinadores* asocian actividades de dedicación a los trabajadores y configuran horarios a los mismos. De este modo, cuando un trabajador rellena la dedicación, podrá selección aquellos días definidos en su horario laboral.

Al rellenar la dedicación, deberemos imputar un número de horas total por día, dicho número de horas es deberá ser alcanzado como mínimo o el sistema generará una *alerta* (Figura 191).

Figura 191. Alerta generada en el sistema por el control de dedicación

El conjunto de dedicaciones permitirá obtener informes en base a dichas inserciones que ayudarán a tomar decisiones en cuanto a la actividad global de la empresa (Figura 192).

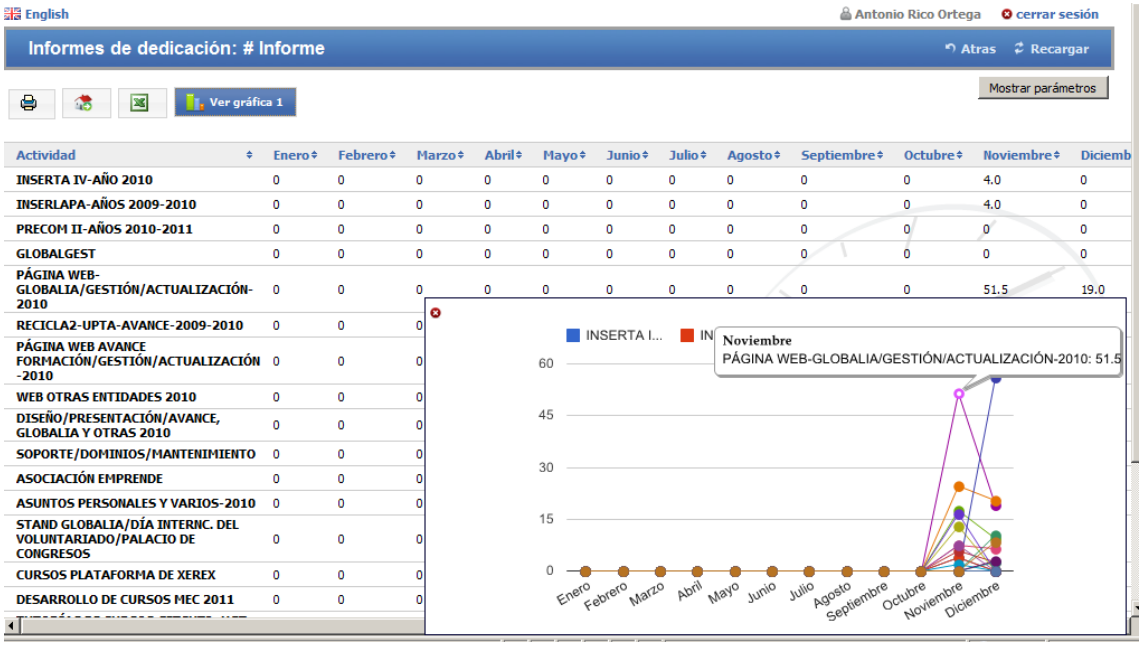


Figura 192. Informe de dedicación de horas a proyectos de los trabajadores

8.2 Control de presencia

Figura 193. Frontend de Control de presencia para la empresa ISD

El control de presencia lleva un registro de las entradas y salidas de los trabajadores. Los trabajadores de la empresa, al entrar y salir de la oficina deben "fichar" en el Frontend (Figura 193). El usuario con permisos oportunos da de alta a los trabajadores

en Globalgest asignándoles una contraseña de acceso que será la que utilicen posteriormente para la entrada y salida del sistema.

Figura 194. Alta de trabajador en el control de presencia

Una vez dado de alta el trabajador deberá indicar sus credenciales en el Frontend cada vez que salga o entre a la oficina. Globalgest mostrará un informe completo de las entradas y salidas de todos los trabajadores (Figura 195).

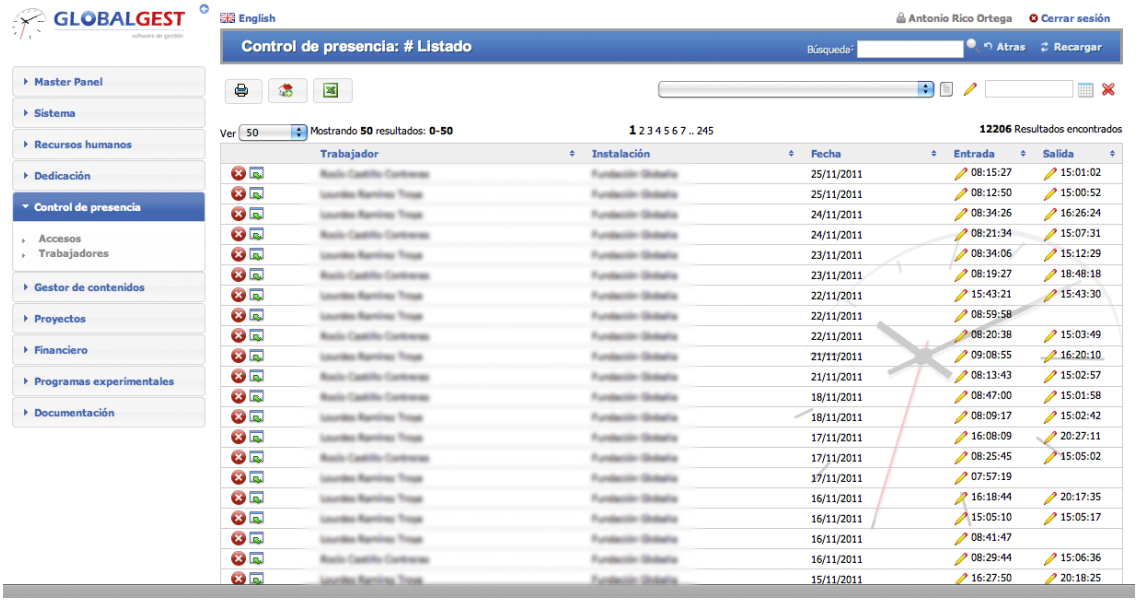


Figura 195. Informe de entradas y salidas de los trabajadores

8.3 Control de Productos

El control de productos es un grupo funcional con seis elementos y mantiene un registro de los productos ofertados por la empresa suscriptora. Vemos una captura de pantalla del menú principal en la Figura 196.

Figura 196. Grupo Funcional Productos en Globalgest

Globalgest establece hasta tres niveles de agrupación (familias, categorías y subcategorías) y permite crear packs de productos con determinadas condiciones. Además existe la posibilidad de asociar a los productos diferentes tarifas.

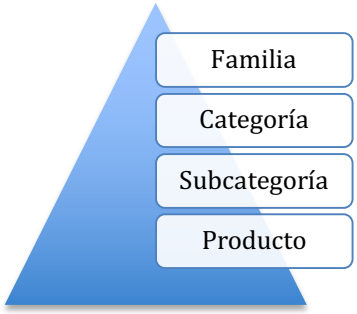


Figura 197. Niveles de agrupación en el control de productos

Así, según ilustra la Figura 197 tendremos *Familias* que se dividen en *Categorías* y estas a su vez en *Subcategorías*. Esta división permite establecer una categorización que será de gran utilidad en la obtención de informes de inteligencia de negocio. Por ejemplo, podremos obtener un desglose de ventas por familia o saber qué subcategoría ha reportado mayor ROI durante la campaña de Navidad.

Un mismo producto podrá tener diferentes tarifas dependiendo de su naturaleza. Por tanto, un mismo producto podrá tener distintos precios. Escoger un precio u otro en el momento de la facturación dependerá de las condiciones concretas del producto o servicio. **Ejemplo** de tarifas: *Tarifa normal, Tarifa mensual*, etc.

Una vez que tenemos configurado el sistema con los niveles de agrupación (familias, categorías y subcategorías), así como las tarificaciones que queremos incluir, la inserción de productos es muy sencilla (Figura 198).

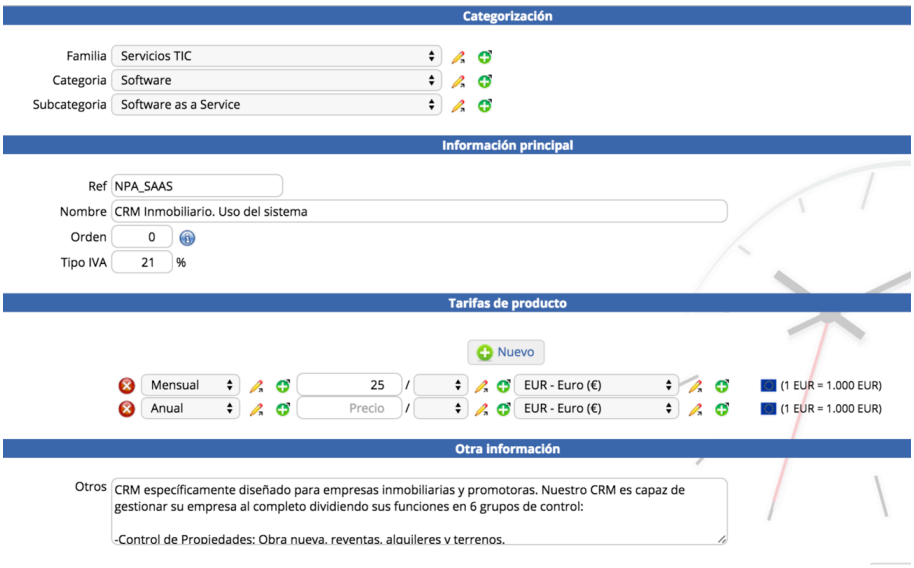


Figura 198. Formulario de inserción de producto en Globalgest

La funcionalidad "*Packs de producto*" permite agrupar varios productos de nuestra cartera con unas determinadas condiciones de venta. Un pack de productos tendrá tarifas determinadas incluyendo todos sus componentes y no se puede separar. La Figura 199 nos muestra un ejemplo de inserción de un registro en esta funcionalidad.

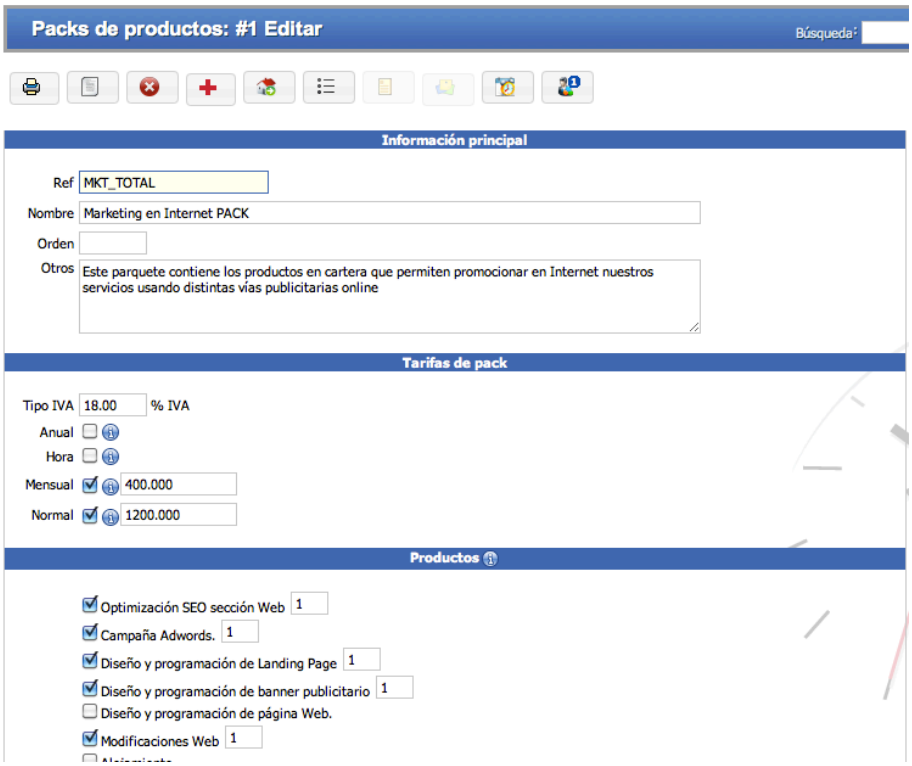


Figura 199. Formulario de inserción de pack de producto

8.4 Control de documentación (DMS)

En un sistema de gestión empresarial debemos distinguir dos tipos de información:

- **Información estructurada**, almacenada en tablas y registros (tabla TT de clientes por ejemplo)
- **Información no estructurada** correspondiente a documentos y material digitalizado que deseamos almacenar.

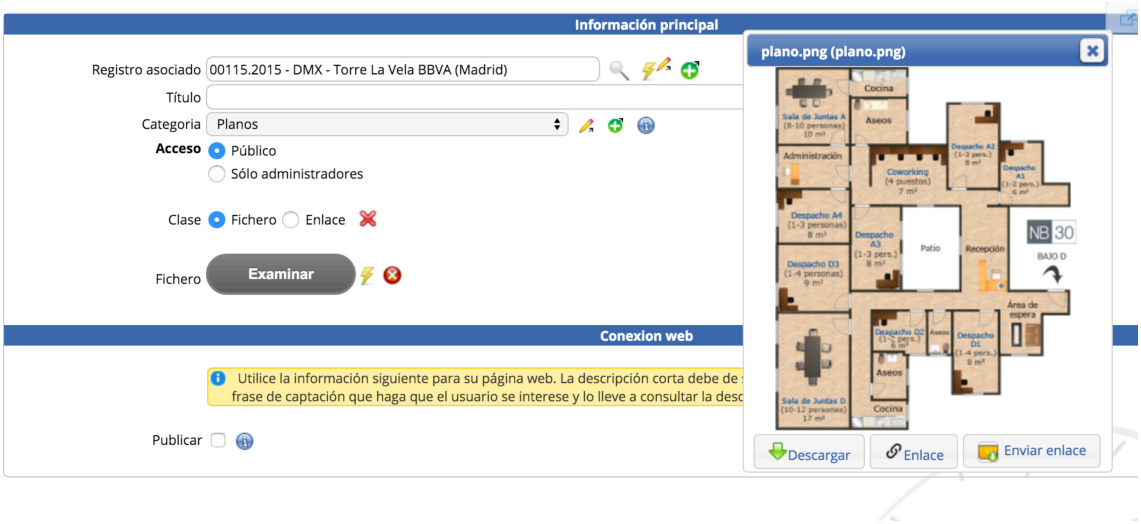


Figura 200 Control de documentación en Globalgest

El control de documentación en Globalgest, permite almacenar información no estructurada en el sistema de gestión de la empresa y acceder a ella de manera segura, sencilla y rápida (Figura 200). Esta funcionalidad de Globalgest la abreviaremos ***DMS***, que viene del Inglés *Document Management System* y quiere decir Sistema de gestión documental. La información documental que almacenemos en Globalgest está *tipificada* esto es, un documento puede tener asociado una categoría, una clase a la que pertenece. No debemos confundir el tipo del documento del formato mime, que se puede extraer del mismo archivo. Supongamos por ejemplo el módulo en Globalgest para el control de proyectos en una empresa constructora; estos registros de proyecto tendrán documentación asociada y es lógico que establezcamos una clasificación, una tipología de los documentos dividiéndolos en: Informes iniciales, planos, fotografías, etc.

Nombre	*Teléfono*
Antonio Rico Ortega	666-666-666
Miguel Rifá Fabregat	666-666-666
Miguel Gómez-Aleixandre	666-666-666
Carlos Torres Díaz	666-666-666
Gema Puertas López	666-666-666

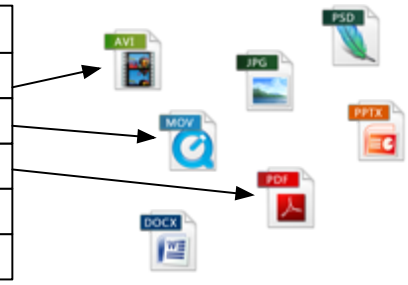


Figura 201. Información estructurada y su asociación a documentos

Los documentos que tengamos en el sistema deben poder asociarse a registros estructurados. De este módulo, para un determinado módulo podemos obtener tanto la información estructurada en las tablas como los documentos asociados al mismo. Tal y como vemos en la Figura 201, determinados documentos aparecen asociados a registros. Otros sin embargo, están en el sistema pero no tienen un registro de TT enlazado.

9 Ejemplo de despliegue funcional: Globalgest *Clínica Médica*

Cada cliente obtiene una implantación personalizada de Globalgest según sus necesidades. Basta con dar de alta las funcionalidades en el panel de control para desplegar el sistema de gestión deseado. Para el caso de la clínica médica seleccionaremos el conjunto de funcionalidades que aparecen en la Figura 202. El despliegue de dichas funcionalidades ya lo hemos vista en la Figura 170 y no es necesario repetirlo.

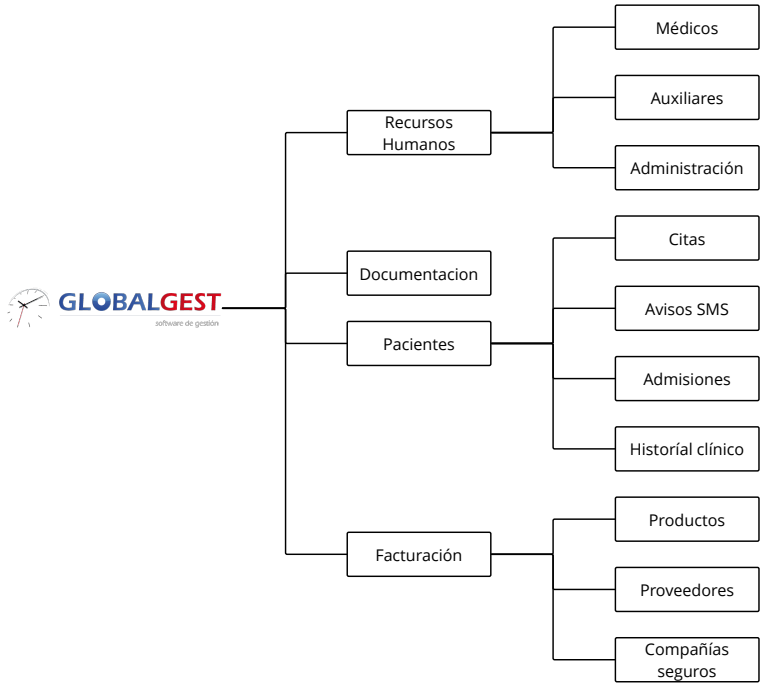


Figura 202. Diagrama de funcionalidades para la implantación de una clínica médica

Tal y como vemos, el sistema de gestión de la clínica presenta muchas funcionalidades comunes que pueden ser reutilizadas en otros sectores: Control de Productos, Facturación, Proveedores, Avisos SMS. Estas funcionalidades son intersectoriales y pueden ser activadas para todos los arrendatarios independientemente del sector al que pertenezcan. Si existen otras, por el contrario, que son desarrolladas específicamente para el cliente.

Así, el *Control de pacientes* permite llevar un registro de los pacientes de la clínica. Dichos pacientes llamarán a la clínica tendrán *citas* para asistir a la clínica un día determinado. La cita será *admisionada* y el paciente pasará a consulta. Una vez finalizada la cita, el personal médico redactará un informe que pasará a forma parte del

historial clínico de paciente. La gestión de citas genera listados de gran utilidad para el personal. Por ejemplo, la generación de un informe con las citas del día y su evolución (Figura 203).

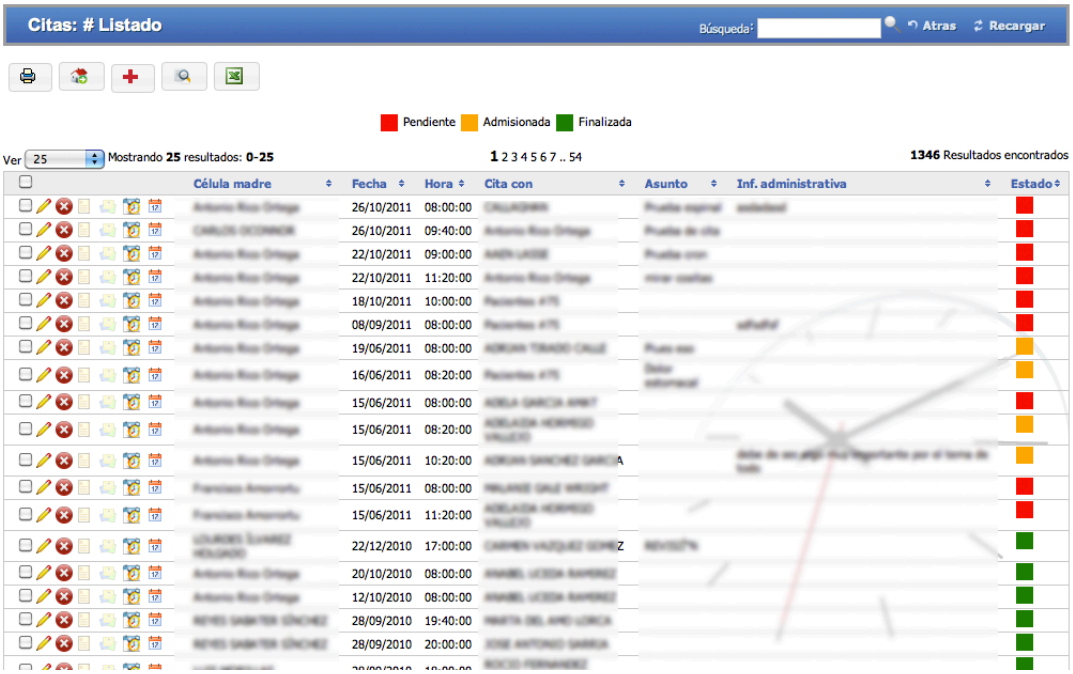


Figura 203. Informe de citas diarias

El sistema alertará mediante recordatorios *SMS* al paciente de la proximidad de la cita. Globalgest ejecuta diariamente tareas CRON en el servidor, entre las cuales se encuentra el envío de recordatorios SMS. En la Figura 204 podemos ver un ejemplo real de un recordatorio.

Figura 204. Recordatorio SMS enviado a paciente

Una vez finalizada la cita, el personal médico redactará un informe que pasará a forma parte del historial clínico de paciente. La factura será emitida bien al paciente de forma directa, o a la compañía aseguradora de la misma.

Este ejemplo de implantación demuestra las ventajas de la reutilización modular. Con tan sólo desarrollar unos módulos específicos de industria, el sistema es capaz de dar servicio al sector médico. El resto de funcionalidades ya están presentes en el sistema y no es necesario desarrollarlas o adaptarlas, basta con "clicarlas".

Chapter XI

Conclusions & future work

We have defined MT^2 as an extension to the current multi-tenant architectures. The main objective set for this thesis has been to foster component reuse in multi-tenancy and permit several functionalities to be served using the same application instance. We feel that the new multi-target resulting approach enables significant benefits to all the actors involved in cloud computing. This chapter summarizes conclusions and future work.

1 Conclusions

MTAs are a key technology for the success of SaaS as a new model for software delivery in cloud computing. Multi-tenancy is an architectural pattern for SaaS applications that 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; this way, SaaS formula is especially attractive to SMEs, as they eliminate capital expenses while operational costs are highly reduced too.

MTAs need an administrative tier to support agile account creation and management. Therefore, multi-tenancy aligns the architecture and opens new paths to the traditional agile stream of ASD; programmers have now a support to the agility in deployment.

However, traditional MT involves different applications and implementations to serve different functionalities (e.g. CRM, ERP, SCM). This basically entails price increase for clients (due to multiple subscriptions) and a narrowed spectrum of potential clients for vendors who can only target those industries or companies related to the functionality deployed (e.g. CRM vendors can only sell to companies needing to improve customer relations). Despite the fact that many components could be quite often shared across different implementations, they are duplicated instead; consequently, developers see their work complicated with unnecessary tasks and the consequent waste of time and money.

Moreover, even though the price reduction that MT provides, SaaS adoption by SMEs is still to come. The range of SMEs is very wide and software necessities for all these companies should not be considered uniformly. In fact, in average four out of five SMEs are micro businesses with less than ten people that either cannot afford SaaS, or applications are so complicated that rather hinder instead of support. Micro vendors on their part have a limited ratio for prospects and cannot risk marketing SaaS applications whose success relies on the scalability.

Therefore, notwithstanding the benefits that traditional MT delivers, it still has drawbacks that are susceptible of study.

The **foundations** for this research are as simple as powerful; there are common components duplicated identically over implantations of applications of different nature. If these elements are functional-independent (programmers do not pay attention on which EIS will be running), why not unify and reuse? Besides, common components are not only reduced to code snippets, we can also include other reusable assets such as presentation layer elements (icons, style sheets, etc.) or database tables (days of the week, countries, etc.). As depicted on Figure 205, MT^2 intends to move from multiple MT implantations to one single MT^2 system encompassing different functionalities that reuse common core components.

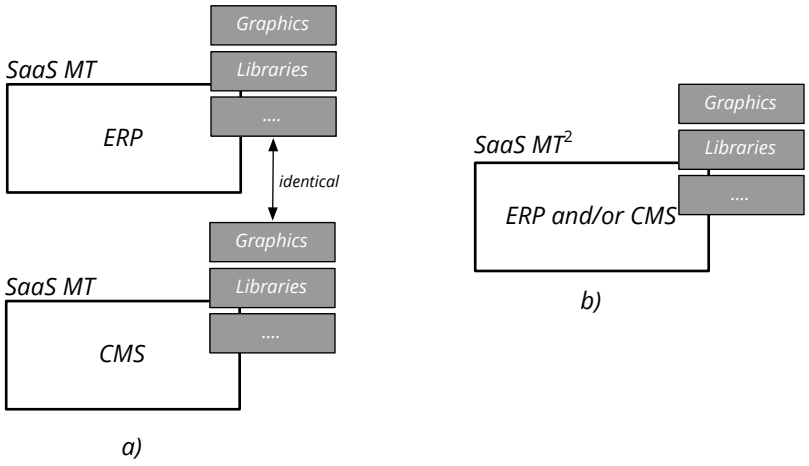


Figure 205.Comparison between MT (a) and MT^2 (b)

Multi-tenancy Multi-target (MT^2) is an approach to extend MT architectures so that **multiple functionalities can be served with the same software instance**. MT^2 unifies a set of MT implementations in one by reusing the common components of the architectures. As stated in our hypothesis, component sharing and reuse over services has solved multi-tenancy disadvantages.

This novel approach enhances MT scalability in both the dimension of tenants and functionalities. With this extension, vendors can have multiple MT systems in single MT^2 application that controls which functionalities to deploy for each tenant. Therefore, the architecture we propose is **not only multi-tenant but also multi-target** as increases the addressable market for vendors.

MT^2 makes the costs of SaaS much lower, plus reducing learning effort and better fitting specific tenants' functional needs. Having a single system to maintain entails an

extra cutting to the already reduced multi-tenant prices. Due to unification, vendors see their costs lowered, and so do the customers. Particularly, SMEs from the bottom end of the spectrum have now the chance to embrace and adopt the cloud. Micro businesses could migrate their legacy applications to affordable MT^2 cloud solutions; an unaffordable option with the traditional multi-tenant applications.

MT^2 **supports agility** not only for deployment but also **for development**. Several MT applications are grouped into a single MT^2 system and used as assets in order to meta-generate customized service applications for clients. MT^2 achieves reusability (and thus supports agility) by removing duplication of common features. With this extension programmers can boost development, since just one application is needed to host many functionalities. Moreover, this unification lessens the tasks of maintenance.

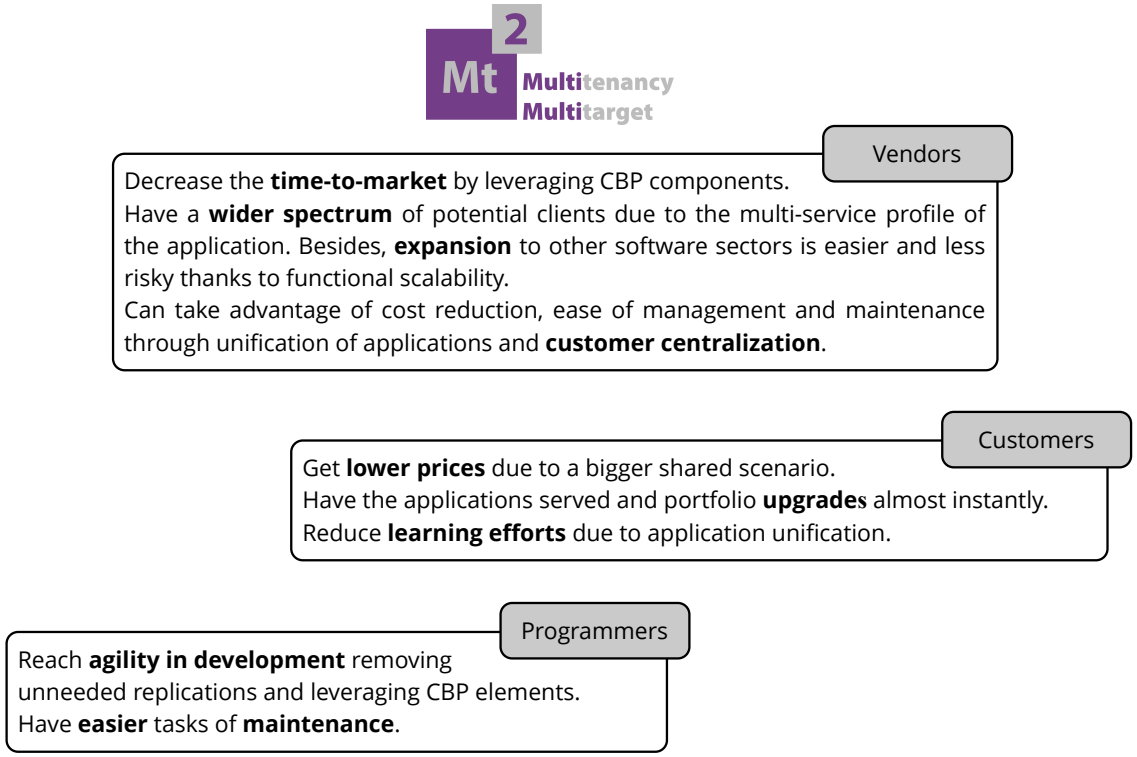


Figure 206. Summary of benefits in SaaS MT^2

To sum up, with this multifunctional situation, **MT^2 solves the inconveniences of the traditional MT architectures** and delivers new benefits to all cloud computing users (Figure 206). Tenants needing functionalities of disparate nature or simply with dissimilar level of complexity will be able to unify services using one single application (with subsequent price reduction and ease of learning effort). Vendors on their side have

the opportunity to broaden the spectrum of potential clients at functional level. In addition, general costs of the use of MT^2 applications are again lowered as so are reduced the costs of the software maintenance and infrastructure needs. Additionally, coders find support to the agility in the development; there will be no need for programmers to duplicate tasks as different functionalities are now hosted under the same operational system.

2 Contributions

Basically, MT^2 turns traditional multi-tenant software into multi- target applications. That means that just one application is needed to serve many different services (ERP, CRM or CMS) on demand. Functionalities are deployed dynamically according to tenants' subscriptions. The main contributions contained herein can be summarized as follows:

- We elicited an architectural framework for traditional MTA models that pays attention to both administrative and instance levels, so far the first contribution for this kind of architectures. We have detailed and illustrated the component-based design and taken it as a base to be extended by our approach.

- We have devised a reference framework for the design of MT databases. This contribution simplifies the architectural efforts in the design of the data layer for multi-tenant environments. This contribution aims to be helpful in future research, as data privacy has been demonstrated to be one of the major concerns in shared scenarios. To this end, we fully reviewed the state-of-art of the MT data layer explaining the different approaches to implement multi-tenancy and schema mapping techniques for tenants' customization.

- This work claims in favor for the coexistence of agility and architecture; the research has demonstrated how architectures can be aligned and used as good assets in software engineering. Particularly, we have shown how MT finds support to the agility in the deployment and how our proposal extends this alignment and support in the agility in the development.

- We have demonstrated the importance of SMEs in world's economy. For this reason, we have made a cared review of types of EIS and studied the factors that affect small companies to embrace the cloud. We have shown how range of SMEs is excessively wide and discussed this same division (based on size) should not be applied at functional requirements level for software applications.

- We have coined "mono-target" as new denomination for the traditional multi-tenancy. We discussed how actual MT applications deploy a single functionality and therefore leave a single target for vendors in the market. Shortcomings and concerns of mono-target systems have been detailed and illustrated.

- We have presented MT^2 as an extension to the traditional mono-target model; a novel approach that enables both selective and multi-functional deployment. Based on reutilization of common components, we have explained how this approach solves the inconveniences of its mono-functional predecessor.

- We have described and detailed an MT^2 architectural model using a component-based design; we have gone through every tier of the architecture, stressing on the modifications and especially caring for the administrative tier and the division of the business layer into the CPB and IBP.

- Based on our contributed framework, we have devised and illustrated a detailed relational model for MT^2 databases, including a novel schema mapping technique called *Pares MT^2 (MT^2 pairs)*. This model can be used as reference in the extension from mono to multi-target systems. Fully detailed relations and examples have been provided.

- We have demonstrated the applicability of the proposal by means of a commercial software application named Globalgest. By using this software, we have been able to:

    - Carry out an experiment to demonstrate the component reuse and multi-target extent of the application; we have also verified the suitability of

the proposal for SMEs and the market increase for vendors due to the reach of despair industries and ease of maintenance.

- Refine mechanisms to perform multi-functional deployment in applications connecting the instance level to the MT^2 metadata configured at administrative level.

- Implement new MT^2 security requirements and levels of. This was done not only at tenant level but also at end user level.

- Split former business layer into CPB and IBP layers. Methodology and examples of how to incorporate statically (CBP) and dynamically (IBP) components have been outlined in this work.

- Introduce and expose some aspects for the administration and support of this kind of multi-functional systems.

3 Future work

The novelty of the approach opens a big field for future studies as the new multi-functional scenario comes with new challenges to overcome. As an example of this, we could consider load balance management: 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 [13], [14] to new servers or hosted alongside functionalities with lower computer needs. The administrative level of the architecture, which needs to incorporate some monitoring and benchmark mechanisms, must deploy this feature.

Coexistence of functionalities with conflictive requirements is also a subject of future research. 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 [201]. For instance, real-time and system flexibility are incompatible by definition. In this regard, recent studies [11], [201] can be used for the analysis and evaluation of candidate EIS

architectures. Considering the new limits of MT^2 scenario, cohabitation might be impossible unless isolation or selection of non-optimal candidates.

Besides incompatibilities, there are other aspects to be considered when providing a multi-functional service: functionalities may be dependent on each other (*appointments* and *agenda* may be dependent to the entities involved like *clients* or *patients* in the case of health industry). This dependency control must be reflected in the MT^2 architecture.

In terms of data layer, we have proposed a model for extending and customizing the base tables; however, this model needs to be implemented and put into practice to see possible shortcomings and improvements applicable.

The rapid growth of the system in terms of tenants must be also 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 [110] or database management systems mechanisms [123].

Internet of Things (IoT) 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 [202], [203].

The MT^2 approach best meets the IT needs of Small and Medium Enterprises (SMEs) and future publications need to study this suitability. Benefits in terms of timesavings for development and deployment should be demonstrated with some empirical data

Bibliography

[1] Q. Zhang, L. Cheng, and R. Boutaba, 'Cloud computing: state-of-the-art and research challenges', *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, Apr. 2010.

[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, 'Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility', *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[3] S. Carlin and K. Curran, 'Cloud Computing Technologies', *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 1, no. 2, pp. 59–65, Jun. 2012.

[4] M. Ambrust, A. Fox, R. Griffith, A. D. A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, 'Above the clouds: A Berkeley view of cloud computing', *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, 2009.

[5] X. Xu, 'From cloud computing to cloud manufacturing', *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 1, pp. 75–86, Feb. 2012.

[6] P. Mell and T. Grance, 'The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology', *NIST Special Publication*, vol. 145, 2011.

[7] A. Prabowo, M. Janssen, and J. Barjis, 'A Conceptual Model for Assessing the Benefits of Software as a Service from Different Perspectives', in *Business Information Systems - 15th International Conference, BIS 2012*, 2012, pp. 108–119.

[8] J. Kaplan, 'Saas: Friend or foe?', *Business Communications Review*, vol. 37, no. June, p. 48, 2007.

[9] A. Rico, M. Noguera, J. L. Garrido, K. Benghazi, and L. Chung, 'Multi-Tenancy Multi-Target (MT2): A SaaS Architecture for the Cloud', in *Advanced Information Systems Engineering Workshops - CAiSE 2012 International Workshops*, 2012, pp. 214–227.

[10] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Second edi. Addison-Wesley Professional, 2003.

[11] N. Niu, L. Da Xu, and Z. Bi, 'Enterprise Information Systems Architecture—Analysis and Evaluation', *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2147–2154, Nov. 2013.

[12] E. J. Qaisar, 'Introduction to cloud computing for developers: Key concepts, the players and their offerings', in *2012 IEEE TCF Information Technology Professional Conference*, 2012, pp. 1–6.

[13] F. Chong and G. Carraro, 'Architecture Strategies for Catching the Long Tail What Is Software as a Service ?', *Most*, vol. 479069, pp. 1–22, 2006.

[14] A. Rico, M. Noguera, J. L. Garrido, K. Benghazi, and L. Chung, 'Supporting Agile Software Development and Deployment in the Cloud : A Multi-tenancy Multi-target Architecture (MT2A)', in *Agile Software Architecture*, M. A. Babar, A. W. Brown, K. Koskimies, and I. Mistrik, Eds. Elsevier, 2013, pp. 269–288.

[15] D. Jacobs and S. Aulbach, 'Ruminations on multi-tenant databases', *Fachtagung fur Datenbanksysteme in Business, Technologie und Web, Aachen, Germany, March*, no. Btw, pp. 5–9, 2007.

[16] A. Rico, M. Noguera, J. L. Garrido, K. Benghazi, and L. Chung, 'Component-Based Design for Multi-tenant Multi-target Support in the Cloud', in *Enterprise and Organizational Modeling and Simulation*, 2013, vol. 153, pp. 146–160.

[17] M. Block, 'Evolving to Agile: A Story of Agile Adoption at a Small SaaS Company', *2011 AGILE Conference*, pp. 234–239, Aug. 2011.

[18] P. Kruchten, 'Software architecture and agile software development: a clash of two cultures?', in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, 2010, vol. 2, pp. 497–498.

[19] P. Abrahamsson, M. A. Babar, and P. Kruchten, 'Agility and Architecture: Can They Coexist?', *IEEE Software*, vol. 27, no. 2, pp. 16–22, Mar. 2010.

[20] J. Madison, 'Agile Architecture Interactions', *Software, IEEE*, vol. 27, no. 2, pp. 41–48, 2010.

[21] G. Booch, 'An Architectural Oxymoron', *Software, IEEE*, vol. 27, no. 5, p. 96, 2010.

[22] C. Anderson, 'The Long Tail: Why the Future of Business Is Selling Less of More by Chris Anderson', *Journal of Product Innovation Management*, vol. 24, no. 3, pp. 1–30, 2007.

[23] F. Chong and G. Carraro, 'Architecture strategies for catching the long tail', *MSDN Library, Microsoft Corporation*, pp. 9–10, 2006.

[24] S. C. Misra and A. Mondal, 'Identification of a company's suitability for the adoption of cloud computing and modelling its corresponding Return on Investment', *Mathematical and Computer Modelling*, vol. 53, no. 3–4, pp. 504–521, Feb. 2011.

[25] D. Catteddu and G. Hogben, 'An SME perspective on Cloud Computing', *Cloud Computing--SME Survey, ENISA report. Online: http://www. enisa. europa. eu/act/rm/files/deliverables/cloud-computing-sme-survey/at_download/fullReport*, 2009.

[26] N. A. Sultan, 'Reaching for the "cloud": How SMEs can manage', *International Journal of Information Management*, vol. 31, no. 3, pp. 272–278, Jun. 2011.

[27] B. Ramdani, P. Kawalek, and O. Lorenzo, 'Predicting SMEs' adoption of enterprise systems', *Journal of Enterprise Information Management*, vol. 22, no. 1/2, pp. 10–24, 2009.

[28] J. Lewandowski, A. O. Salako, and A. Garcia-Perez, 'SaaS Enterprise Resource Planning Systems: Challenges of Their Adoption in SMEs', *2013 IEEE 10th International Conference on e-Business Engineering*, pp. 56–61, Sep. 2013.

[29] F. T. Neves, F. C. Marta, A. M. Ramalho, R. Correia, and M. de C. Neto, 'The Adoption of Cloud Computing by SMEs : Identifying and Coping with External Factors', *11ª Conferência da Associação Portuguesa de Sistemas de Informação (CAPSI)*, no. Capsi 2011, p. 11, 2011.

[30] J. R. King, 'The Challenge of the Computer Utility', *Journal of the Operational Research Society*, vol. 18, no. 3. Addison-Wesley Educational Publishers Inc, pp. 324–325, 1967.

[31] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and others, 'A view of cloud computing', *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[32] Hewlett-Packard, 'Five myths of cloud computing', *Business white paper*, May-2011. [Online]. Available: http://www.hp.com/hpinfo/newsroom/press_kits/2011/HPDiscover2011/DISCOVER_5_Myths_of_Cloud_Computing.pdf.

[33] P. Mell and T. Grance, 'The NIST definition of cloud computing', *NIST Special Publication 800-145*, no. September, p. 7, 2011.

[34] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, 'Scientific cloud computing: Early definition and experience', in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, 2008, pp. 825–830.

[35] A. Rosenthal, P. Mork, M. H. Li, J. Stanford, D. Koester, and P. Reynolds, 'Cloud computing: a new business paradigm for biomedical information sharing.', *Journal of biomedical informatics*, vol. 43, no. 2, pp. 342–53, Apr. 2010.

[36] L. Wang, G. Von Laszewski, M. Kunze, and J. Tao, 'Cloud computing : A Perspective study', *Proceedings of the Grid Computing Environments (GCE) workshop*, no. November, 2008.

[37] E. A. N. da Silva and D. Lucrédio, 'Software engineering for the cloud: A research roadmap', in *Software Engineering (SBES), 2012 26th Brazilian Symposium on*, 2012, pp. 71–80.

[38] R. Miralles, 'Cloud computing y protección de datos', vol. 11, pp. 14–23, 2010.

[39] C.-P. Bezemer and A. Zaidman, '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*, 2010, p. 88.

[40] W. Vogels, 'A Head in the Cloud - The Power of Infrastructure as a Service',

First workshop on Cloud Computing and in Applications CCA 08, pp. 1–29, 2008.

[41] L. Vaquero and L. Rodero-Merino, 'A break in the clouds: towards a cloud definition', *ACM SIGCOMM*, vol. 39, no. 1, pp. 50–55, 2008.

[42] J. Pyke, 'Now is the time to take the cloud seriously', *Cordys White Paper*, 2011. [Online]. Available: http://www.cordysprocessfactory.com/sites/default/files/downloads/Time_to_take_the_cloud_seroiusly_online.pdf. [Accessed: 17-Apr-2013].

[43] Á. Á. H. Bravo, 'El SaaS y el Cloud-Computing: una opción innovadora para tiempos de crisis', *REICIS. Revista Española de Innovación, Calidad e Ingeniería del Software*, vol. 5, no. 1, pp. 38–41, 2010.

[44] H. R. Motahari-Nezhad, B. Stephenson, and S. Singhal, 'Outsourcing business to cloud computing services: Opportunities and challenges', *IEEE Internet Computing*, vol. 10, 2009.

[45] 'Amazon Elastic Compute Cloud (Amazon EC2).' [Online]. Available: http://aws.amazon.com/es/ec2/. [Accessed: 11-Aug-2015].

[46] 'Google Compute Engine - Cloud Computing & Infrastructure As A Service — Google Cloud Platform — Google Cloud Platform.' [Online]. Available: https://cloud.google.com/products/compute-engine/. [Accessed: 10-Aug-2014].

[47] 'Microsoft Azure.' [Online]. Available: http://www.windowsazure.com/en-us/. [Accessed: 11-Dec-2011].

[48] 'Rackspace - Líder en Managed Cloud y Dedicated IaaS.' [Online]. Available: http://www.rackspace.com/es/. [Accessed: 10-Aug-2014].

[49] 'Gartner. Toolkit: Comparison Matrix for Cloud Infrastructure as a Service Providers, 2013.' [Online]. Available: https://www.gartner.com/doc/2575815. [Accessed: 10-Aug-2014].

[50] 'Dedicated Cloud - OVH.' [Online]. Available: https://www.ovh.es/dedicated-cloud/. [Accessed: 02-Nov-2015].

[51] 'Google App Engine - Google Code.' [Online]. Available: https://appengine.google.com/. [Accessed: 11-Aug-2015].

[52] 'Force.com.' [Online]. Available: http://www.salesforce.com/platform/. [Accessed: 12-Aug-2015].

[53] 'SalesForce.com.' [Online]. Available: http://www.salesforce.com/. [Accessed: 12-Aug-2011].

[54] 'Google Drive.' [Online]. Available: https://drive.google.com. [Accessed: 10-Aug-2014].

[55] 'Dropbox.' [Online]. Available: http://www.dropbox.com/. [Accessed: 22-Aug-

2014].

[56] ‘Amazon Simple Storage Service (Amazon S3).’ [Online]. Available: http://aws.amazon.com/es/s3/. [Accessed: 15-Aug-2014].

[57] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, ‘Multi-tenant databases for software as a service’, in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD ’08*, 2008, p. 1195.

[58] ‘Amazon SimpleDB.’ [Online]. Available: http://aws.amazon.com/es/simpledb/. [Accessed: 12-Aug-2015].

[59] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, ‘Bigtable’, *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, Jun. 2008.

[60] G. Lars, ‘HBase Storage Architecture’, 2009. [Online]. Available: http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html. [Accessed: 13-Aug-2013].

[61] Datadog, ‘Datadog - Monitoring as a Service.’ [Online]. Available: https://www.datadoghq.com/. [Accessed: 10-Aug-2014].

[62] ‘Cloudnexa » Cloud Management as a Service.’ [Online]. Available: http://www.cloudnexa.com/service-offerings/cmaas/. [Accessed: 10-Apr-2014].

[63] ‘AppExchange - Hundreds of Cloud Computing Applications, Pre-Integrated with salesforce.com.’ [Online]. Available: http://appexchange.salesforce.com/home. [Accessed: 12-Aug-2013].

[64] ‘vmware españa: virtualización vmware de escritorios, servidores y máquinas virtuales.’ [Online]. Available: http://www.vmware.com/es/. [Accessed: 12-Aug-2014].

[65] ‘xen.org, home of the Xen® hypervisor, the powerful open source industry standard for virtualization.’ [Online]. Available: http://www.xenproject.org/. [Accessed: 12-Aug-2014].

[66] A. Rico, M. Noguera, J. L. Garrido, K. Benghazi, and J. Barjis, ‘Extending multi-tenant architectures: a database model for a multi-target support in SaaS applications’, *Enterprise Information Systems*, pp. 1–22, Sep. 2014.

[67] ‘SOAP Specifications.’ [Online]. Available: http://www.w3.org/TR/soap/. [Accessed: 12-Aug-2011].

[68] ‘Web Service Definition Language (WSDL).’ [Online]. Available: http://www.w3.org/TR/wsdl. [Accessed: 12-Aug-2015].

[69] ‘ Wikimedia Foundation.’ [Online]. Available: http://wikimediafoundation.org/wiki/Fundraising. [Accessed: 11-Dec-2011].

[70] ‘AdSense – Anuncios Google.’ [Online]. Available:

http://www.google.com/adsense/start/. [Accessed: 11-May-2015].

[71] 'DoubleClick.' [Online]. Available: https://www.google.es/intl/es/doubleclick/. [Accessed: 11-May-2015].

[72] J. Gray, 'A Conversation with Jim Gray', *ACM Queue*, vol. 1, no. 4, pp. 8–17, 2003.

[73] D. G. Feitelson and L. Rudolph, 'Gang scheduling performance benefits for fine-grain synchronization', *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, pp. 306–318, 1992.

[74] G. Liu, H. Jiang, and R. Geng, 'Software design on a SaaS platform', *2010 2nd International Conference on Computer Engineering and Technology*, pp. V4–355–V4–358, 2010.

[75] J. O. Coplien and T. Reenskaug, 'The DCI Paradigm: Taking Object Orientation into the Architecture World', in *Agile Software Architecture: Aligning Agile Processes and Software Architectures*, 2013, pp. 25–62.

[76] P. Brereton, D. Budgen, K. Bennnett, M. Munro, P. Layzell, L. MaCaulay, D. Griffiths, and C. Stannett, 'The future of software', *Commun. ACM*, vol. 42, no. 12, pp. 78–84, 1999.

[77] M. Turner, D. Budgen, and P. Brereton, 'Turning software into a service', *Computer*, vol. 36, no. 10, pp. 38–44, Oct. 2003.

[78] C. Anderson, 'The Long Tail', *Wired Magazine*, 2004. [Online]. Available: http://archive.wired.com/wired/archive/12.10/tail.html. [Accessed: 11-Aug-2014].

[79] M. Papazoglou, 'Service-oriented computing: concepts, characteristics and directions', in *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat. No.03CH37417)*, 2003, pp. 3–12.

[80] C. Souza, 'Software as a Service (Saas) vs the ASP Model.' [Online]. Available: http://ezinearticles.com/?Software-as-a-Service-(Saas)-vs-the-ASP-Model&id=637889. [Accessed: 11-Aug-2014].

[81] A. Benlian and T. Hess, 'Opportunities and risks of software-as-a-service: Findings from a survey of IT executives', *Decision Support Systems*, vol. 52, no. 1, pp. 232–246, Dec. 2011.

[82] T. Mäkilä, A. Järvi, M. Rönkkö, and J. Nissilä, 'How to Define Software-as-a-Service--An Empirical Study of Finnish SaaS Providers', in *Software Business*, Springer, 2010, pp. 115–124.

[83] M. Sääksjärvi, A. Lassila, and H. Nordström, 'Evaluating the software as a service business model: From CPU time-sharing to online innovation sharing', *Perspective*, pp. 177–186, 2005.

[84] A. Patel, A. Seyfi, Y. Tew, and A. Jaradat, 'Comparative study and review of

grid, cloud, utility computing and software as a service for use by libraries', *Library Hi Tech News*, vol. 28, no. 3, pp. 25–32, 2011.

[85] H. Liao, 'SaaS business model for software enterprise', in *2010 2nd IEEE International Conference on Information Management and Engineering*, 2010, pp. 604–607.

[86] 'Google+ API - Google+ Platform — Google Developers.' [Online]. Available: https://developers.google.com/+/api/. [Accessed: 13-Aug-2014].

[87] L. Fink and S. Markovich, 'Generic verticalization strategies in enterprise system markets: An exploratory framework', *Journal of Information Technology*, vol. 23, no. 4, pp. 281–296, Oct. 2008.

[88] Jan Sysmans, 'Software-as-a-Service; A Comprehensive Look at the Total Cost of Ownership of Software Applications', *Software & Information Industry Association*, no. September, 2006.

[89] J. F. Dippenaar and R. Butler, 'Software-as-a-service (SaaS): Considerations and implications for SaaS customers', Stellenbosch: Stellenbosch University, 2008.

[90] M. Hogan, F. Liu, A. Sokol, and J. Tong, 'Nist cloud computing standards roadmap', *NIST Special Publication*, vol. 35, 2011.

[91] C. Bezemer and A. Zaidman, 'Challenges of Reengineering into Multi-Tenant SaaS Applications', Delft University of Technology, Tech. Rep. TUD-SERG-2010-012, 2010.

[92] M. Almorsy, J. Grundy, and A. Ibrahim, 'SMURF: Supporting Multi-tenancy Using Re-aspects Framework', in *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*, 2012, pp. 361–370.

[93] J. Kr, S. Enderlein, M. Helmich, and A. Zeier, 'Customizing Enterprise Software as a Service Applications : Back-End Extension in a Multi-tenancy Environment', *Work*, vol. 24, no. I, pp. 66–77, 2009.

[94] C. Low, Y. Chen, and M. Wu, 'Understanding the determinants of cloud computing adoption', *Industrial Management & Data Systems*, vol. 111, no. 7, pp. 1006–1023, 2011.

[95] N. Pramod, A. K. Muppalla, and K. G. Srinivasa, 'Limitations and Challenges in Cloud-Based Applications Development', in *Software Engineering Frameworks for the Cloud Computing Paradigm*, Z. Mahmood and S. Saeed, Eds. London: Springer London, 2013, pp. 55–75.

[96] B. R. Kandukuri, R. P. V., and A. Rakshit, 'Cloud Security Issues', *2009 IEEE International Conference on Services Computing*, pp. 517–520, 2009.

[97] W3C, 'Web Accessibility Initiative (WAI) - home page.' [Online]. Available: http://www.w3.org/WAI/. [Accessed: 18-Aug-2014].

[98] Wikipedia, 'Larga cola - Wikipedia, la enciclopedia libre.' [Online]. Available:

http://es.wikipedia.org/wiki/Larga_cola. [Accessed: 17-Aug-2011].

[99] Babalum (Traducción Wired), 'La Larga Estela – El fin de Pareto', 2006. [Online]. Available: http://babalum.com/2006/10/12/la-larga-estela-el-fin-de-pareto/. [Accessed: 17-Aug-2011].

[100] D. (IBM G. B. S. Lashar J, 'To SaaS or not to SaaS?', *destinationcrm.com*, 2009. [Online]. Available: http://www.destinationcrm.com/Articles/Columns-Departments/The-Tipping-Point/To-SaaS-or-Not-to-SaaS-53686.aspx. [Accessed: 16-Aug-2011].

[101] D. Jacobs, 'Enterprise software as service', *Queue*, vol. 3, no. 6, p. 36, Jul. 2005.

[102] C. Anderson, *The long tail: how endless choice is creating unlimited demand*. Random House, 2007.

[103] F. Chong, G. Carraro, R. Wolter, M. Corporation, and A. Architecture, 'Multi-Tenant Data Architecture Three Approaches to Managing Multi-Tenant Data', *Architecture*, vol. 479086, no. June, pp. 1–18, 2006.

[104] J. Varia, 'Architecting for the cloud: Best practices', *Amazon Web Services*, vol. 1, no. January, pp. 1–21, 2010.

[105] 'Un rayo tumba las webs de Menéame, Paypal y Amazon.' [Online]. Available: http://www.meneame.net/story/rayo-tumba-webs-meneame-paypal-amazon. [Accessed: 16-Aug-2011].

[106] P. Coffee, 'Busting Myths of On-Demand : Why Multi-Tenancy Matters', *Salesforce.com (whitepaper)*, 2007. [Online]. Available: https://developer.salesforce.com/page/File:MythbustMultiT.PDF.

[107] D. Banks, J. Erickson, and M. Rhodes, 'Multi-tenancy in cloud-based collaboration services', *Hewlett-Packard Development Company, LP*, 2009. [Online]. Available: http://www.hpl.hp.com/techreports/2009/HPL-2009-17.pdf. [Accessed: 14-Jul-2013].

[108] W.-W. Wu, L. W. Lan, and Y.-T. Lee, 'Exploring decisive factors affecting an organization's SaaS adoption: A case study', *International Journal of Information Management*, vol. 31, no. 6, pp. 556–563, Dec. 2011.

[109] I. Son, D. Lee, J.-N. Lee, and Y. B. Chang, 'Market Perception on Cloud Computing Initiatives in Organizations: An Extended Resource-based View', *Information & Management*, vol. 51, no. 6, pp. 653–669, Jun. 2014.

[110] X. Li, T. Liu, Y. Li, and Y. Chen, 'SPIN: Service performance isolation infrastructure in multi-tenancy environment', *Service-Oriented Computing–ICSOC 2008*, pp. 649–663, 2008.

[111] G. Shroff, 'Multi-tenant software', in *Enterprise Cloud Computing - Technology, Architecture, Applications.*, Cambridge University Press, 2010, pp. 104–114.

[112] C.-F. Liao, K. Chen, and J.-J. Chen, 'Modularizing tenant-specific schema

customization in SaaS applications', in *Proceedings of the 8th International Workshop on Advanced Modularization Techniques - AOAsia'13*, 2013, pp. 9–12.

[113] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold, 'A comparison of flexible schemas for software as a service', in *Proceedings of the 35th SIGMOD international conference on Management of data - SIGMOD '09*, 2009, pp. 881–888.

[114] C. D. Weissman and S. Bobrowski, '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*, 2009, pp. 889–896.

[115] Microsoft, 'Sparse Columns', 2008. [Online]. Available: http://technet.microsoft.com/en-us/library/ff427239.aspx.

[116] G. P. Copeland and S. N. Khoshafian, 'A decomposition storage model', in *ACM SIGMOD Record*, 1985, vol. 14, no. 4, pp. 268–279.

[117] L. S. Q. L. Xml, W.-J. Chen, and F. Hendricks, 'DB2 9 pureXML Guide', *Contract*, 2006.

[118] C. M. Saracca, D. Chamberlin, and R. Ahuja, *DB2 9: pureXML Overview and Fast Start*. IBM Redbooks, 2006.

[119] F. S. Foping, I. M. Dokas, J. Feehan, and S. Imran, 'A new hybrid schema-sharing technique for multitenant applications', in *Digital Information Management, 2009. ICDIM 2009. Fourth International Conference on*, 2009, pp. 1–6.

[120] Apache, 'HBase.' [Online]. Available: http://hbase.apache.org/. [Accessed: 13-Aug-2013].

[121] A. S. Aiyer, M. Bautin, G. J. Chen, P. Damania, P. Khemani, K. Muthukkaruppan, K. Ranganathan, N. Spiegelberg, L. Tang, and M. Vaidya, 'Storage Infrastructure Behind Facebook Messages: Using HBase at Scale.', *IEEE Data Eng. Bull.*, vol. 35, no. 2, pp. 4–13, 2012.

[122] H. Yaish, M. Goyal, and G. Feuerlicht, 'An Elastic Multi-tenant Database Schema for Software as a Service', in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, 2011, pp. 737–743.

[123] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An, '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*, 2008, pp. 94–101.

[124] L. Da Xu, 'Enterprise Systems: State-of-the-Art and Future Trends', *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 630–640, Nov. 2011.

[125] S. Li, L. Xu, X. Wang, and J. Wang, 'Integration of hybrid wireless networks in

cloud services oriented enterprise information systems', *Enterprise Information Systems*, vol. 6, no. 2, pp. 165–187, May 2012.

[126] T. H. Davenport, 'Putting the enterprise into the enterprise system.', *Harvard business review*, vol. 76, no. 4, pp. 121–31, 1998.

[127] M. L. Markus and C. Tanis, 'The enterprise systems experience–from adoption to success', *Framing the Domains of IT Management: Projecting the Future Through the Past*, vol. 173, pp. 173–207, 2000.

[128] J. W. Ross and M. R. Vitale, 'The ERP Revolution: Surviving vs. Thriving', *Information Systems Frontiers*, vol. 2, no. 2, pp. 233–241, 2000.

[129] G. Shroff, *Enterprise cloud computing: technology, architecture, applications*. Cambridge university press, 2010.

[130] K. B. Hendricks, V. R. Singhal, and J. K. Stratman, 'The impact of enterprise systems on corporate performance: A study of ERP, SCM, and CRM system implementations', *Journal of Operations Management*, vol. 25, no. 1, pp. 65–82, Jan. 2007.

[131] G. Shroff, 'Enterprise software: ERP, SCM, CRM', in *Enterprise Cloud Computing Technology, Architecture, Applications*, 2010, pp. 161–177.

[132] H. Klaus, M. Rosemann, and G. Gable, 'What is ERP?', *Information systems frontiers*, vol. 2, no. 2, p. 141, 2000.

[133] M. L. Markus, C. Tanis, and P. C. van Fenema, 'MULTISITE ERP IMPLEMENTATIONS.', *Communications of the ACM*, vol. 43, no. 4, pp. 42–46, 2000.

[134] G. Sch, M. Schulze, Y. Yusuf, and A. Musa, 'The Benefits of SaaS-Based Enterprise Systems for SMEs - A Literature Review', pp. 34–45, 2013.

[135] S. España, 'Software de Gestion y Estrategia SAP | Soluciones y programas de gestion empresarial.' [Online]. Available: http://www.sap.com/spain/index.epx. [Accessed: 24-Aug-2011].

[136] SAP, 'SAP at a Glance. The SAP Capital Market Story', *SAP Financial Report*, 2013.

[137] I. J. Chen and K. Popovich, 'Understanding customer relationship management (CRM): People, process and technology', *Business Process Management Journal*, vol. 9, no. 5, pp. 672–688, 2003.

[138] A. Rico, 'NetPropertyAgent - SaaS Real Estate CRM', *desarrollotic.com*. [Online]. Available: http://www.netpropertyagent.com/. [Accessed: 17-Nov-2012].

[139] H. Stadtler, 'Supply chain management and advanced planning—basics, overview and challenges', *European Journal of Operational Research*, vol. 163, no. 3, pp. 575–588, Jun. 2005.

[140] R. D. Ireland, R. Hoskisson, and M. Hitt, *Understanding Business Strategy: Concepts and Cases*. Cengage Learning, 2008.

[141] 'Drupal - Open Source CMS | drupal.org.' [Online]. Available: http://drupal.org/. [Accessed: 24-Aug-2013].

[142] 'Joomla!' [Online]. Available: http://www.joomla.org/. [Accessed: 24-May-2015].

[143] 'AIIM - The Global Community of Information Professionals.' [Online]. Available: http://www.aiim.org/. [Accessed: 25-Aug-2011].

[144] M. Ayyagari, T. Beck, and A. Demirguc-Kunt, 'Small and Medium Enterprises Across the Globe', *Small Business Economics*, vol. 29, no. 4, pp. 415–434, Jan. 2007.

[145] K. K. Kobe and E. C. Services, *The small business share of GDP, 1998-2004*, no. April. SBA Office of Advocacy, 2007.

[146] M. D. Griffiths, L. Gundry, J. Kickul, and A. M. Fernandez, 'Innovation ecology as a precursor to entrepreneurial growth: A cross-country empirical investigation', *Journal of Small Business and Enterprise Development*, vol. 16, no. 3, pp. 375–390, 2009.

[147] E. Kommission, *The new SME definition: User guide and model declaration*. European Comm., Publication Office, 2005.

[148] P. Wymenga, V. Spanikova, A. Barker, J. Konings, and E. Canton, 'EU SMEs in 2012: at the crossroads', *Annual report on small and medium-sized enterprises in the EU*, vol. 12, no. September, 2011.

[149] 'Small and Medium-Sized Enterprises: Characteristics and Performance', *Investigation No. 332-510. USITC Publication 4189*, no. 332, p. 228, 2010.

[150] U.S. Department of Commerce, 'Statistics of U.S. Businesses.' [Online]. Available: http://www.census.gov/econ/susb/. [Accessed: 13-May-2013].

[151] E. Lukács, 'The economic role of SMEs in world economy, especially in Europe', *European Integration Studies*, vol. 4, no. 1, pp. 3–12, 2005.

[152] C. Lima, C. Econ, C. Aut, L. Pymes, L. Administraciones, D. Local, and I. Locales, 'LAS PYMES Y EL DESARROLLO LOCAL : UNA ALTERNATIVA PARA LA ECONOMÍA ANDALUZA', pp. 253–260.

[153] www.ipyme.org, 'Estadísticas PYME - Evolución e Indicadores (Ministerio de Industria, Energía y Turismo)', 2013. [Online]. Available: http://www.ipyme.org/Publicaciones/Estadisticas-Pyme-n11-Marzo-2013.pdf.

[154] S. Lee, S. B. Park, and G. G. Lim, 'Using balanced scorecards for the evaluation of "Software-as-a-service"', *Information & Management*, vol. 50, no. 7, pp. 553–561, Nov. 2013.

[155] B.-K. Jeong and A. C. Stylianou, 'Market reaction to application service provider (ASP) adoption: An empirical investigation', *Information & Management*, vol. 47, no. 3, pp. 176–187, Apr. 2010.

[156] SAP, 'Consulting Areas and Consultant Levels. Rates and Fees', *SAP Nederland consulting services*, 2011. [Online]. Available: http://docs.desarrollotic.com/tesis_documentacion/SAP - 2011 - Consulting Areas and Consultant Levels. Rates and Fees.pdf.

[157] T. Oliveira, M. Thomas, and M. Espadanal, 'Assessing the determinants of cloud computing adoption: An analysis of the manufacturing and services sectors', *Information & Management*, vol. 51, no. 5, pp. 497–510, Jul. 2014.

[158] T. Haselmann and G. Vossen, 'Software-as-a-service in small and medium enterprises: an empirical attitude assessment', in *Web Information System Engineering--WISE 2011*, no. 1, Springer, 2011, pp. 43–56.

[159] R. Macasaet, L. Chung, J. L. Garrido, M. Noguera, and M. L. Rodríguez, 'An agile requirements elicitation approach based on NFRs and business process models for micro-businesses', in *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement - Profes '11*, 2011, p. 50.

[160] C. Rettig, 'The Trouble with Enterprise Software', *MIT Sloan Management Review*, vol. 49, no. 49101, pp. 21–27, 2007.

[161] M. R. Karabek, J. Kleinert, and A. Pohl, 'Cloud Services for SMEs – Evolution or Revolution?', *Business + Innovation*, vol. 2, no. 1, pp. 26–33, Feb. 2011.

[162] L. Williams, 'What agile teams think of agile principles', *Communications of the ACM*, vol. 55, no. 4, p. 71, Apr. 2012.

[163] S. Liu, Y. Zhang, and X. Meng, 'Towards High Maturity in SaaS Applications Based on Virtualization', *International Journal of Information Systems in the Service Sector*, vol. 3, no. 4, pp. 39–53, 2011.

[164] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, 'Manifesto for Agile Software Development', *The Agile Alliance*, 2001. [Online]. Available: http://agilemanifesto.org/. [Accessed: 28-Nov-2011].

[165] K. Beck, *Extreme programming explained: embrace change*, vol. 92, no. c. Addison-Wesley Professional, 2000.

[166] 'Agile Alliance :: Home.' [Online]. Available: http://www.agilealliance.org/. [Accessed: 29-Nov-2011].

[167] J. H. Canós, P. Letelier, C. Penadés, and D. P. De Valencia, 'Metodologías Ágiles en el Desarrollo de Software', *Development*, pp. 1–8.

[168] M. Poppendieck and T. Poppendieck, *Lean software development: An agile*

toolkit. Addison-Wesley Professional, 2003.

[169] V. Szalvay, 'An introduction to Agile software development', *Danube Technologies*, no. June, pp. 1–9, 2004.

[170] D. Bishop and A. Deokar, 'Toward an Understanding of Preference for Agile Software Development Methods from a Personality Theory Perspective', *2014 47th Hawaii International Conference on System Sciences*, pp. 4749–4758, 2014.

[171] M. Laanti, O. Salo, and P. Abrahamsson, 'Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation', *Information and Software Technology*, vol. 53, no. 3, pp. 276–290, Mar. 2011.

[172] E. H. Uribe and L. E. V. Ayala, 'Del Manifiesto Ágil, sus valores y sus principios', *redalyc.uaemex.mx*, no. 34, pp. 381–385, 2007.

[173] K. Beck, 'Embracing change with extreme programming', *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

[174] 'Extreme Programming: A Gentle Introduction.' [Online]. Available: http://www.extremeprogramming.org/. [Accessed: 22-Aug-2014].

[175] P. Letelier and C. Penadés, 'Metodologías ágiles para el desarrollo de software : eXtreme Programming (XP)', *Development*.

[176] R. Jeffries, A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. Addison-Wesley, 2001.

[177] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, vol. 18. Prentice Hall, 2001.

[178] 'Adaptative Project Management Using Scrum.' [Online]. Available: http://www.methodsandtools.com/archive/archive.php?id=18. [Accessed: 10-May-2014].

[179] A. Cockburn, 'Agile Software Development', *Computer*, vol. 34, no. 9, pp. 120–127, 2001.

[180] J. Stapleton, *DSDM: Dynamic Systems Development Method: The Method in Practice*. Addison Wesley, 1997.

[181] J. Highsmith, 'Adaptive Software Development', *Adaptive software development*, pp. 173–179, 1999.

[182] F.-D. Development, 'Feature-Driven Development', *Java Modeling in Color with UML*, pp. 182–203, 1999.

[183] R. M. Becker, 'Lean manufacturing and the Toyota production system', *Encyclopedia of World Biography*, 1998.

[184] O. Cawley, X. Wang, and I. Richardson, 'Lean/agile software development methodologies in regulated environments–state of the art', *Lean Enterprise*

Software and Systems, pp. 2–7, 2010.

[185] R. Charette, 'Software Intensive Systems Lean Development (ITAHBI Corporation).' [Online]. Available: http://www.itabhi.com/ld.htm. [Accessed: 01-Dec-2011].

[186] J. Martin, 'Rapid application development', *History*, no. 301, p. 788, 1991.

[187] W. W. Royce, 'Managing the development of large software systems', in *proceedings of IEEE WESCON*, 1970, vol. 26, no. 8, pp. 328–388.

[188] G. K. Hanssen and T. E. Fægri, 'Process fusion: An industrial case study on agile software product line engineering', *Journal of Systems and Software*, vol. 81, no. 6, pp. 843–854, 2008.

[189] K. Pohl, G. Böckle, and F. Van Der Linden, *Software Product Line Engineering. Foundations, Principles, and Techniques*, vol. 49, no. 12. Springer-Verlag New York Inc, 2005.

[190] P. Clements and J. McGregor, 'Better, faster, cheaper: Pick any three', *Business Horizons*, vol. 55, no. 2, pp. 201–208, 2012.

[191] 'Sales Cloud: Sales Force Automation Tools - Salesforce.com.' [Online]. Available: https://www.salesforce.com/sales-cloud/overview/. [Accessed: 19-May-2015].

[192] CrownPeak Technology, 'Crownpeak: SaaS Web Content Management, Cloud CMS, Enterprise CMS.'

[193] 'Software para organización de eventos online - amiando.es.' [Online]. Available: https://www.amiando.com/?requestLanguage=ES. [Accessed: 10-Nov-2011].

[194] 'Business Solutions | Solutions | Oracle.' [Online]. Available: http://www.oracle.com/us/solutions/business-solutions-444942.html. [Accessed: 29-Aug-2015].

[195] Q. Li, J. Zhou, Q.-R. Peng, C.-Q. Li, C. Wang, J. Wu, and B.-E. Shao, 'Business processes oriented heterogeneous systems integration platform for networked enterprises', *Computers in Industry*, vol. 61, no. 2, pp. 127–144, Feb. 2010.

[196] Q. Li, Z. Wang, W. Li, Z. Cao, R. Du, and H. Luo, 'Model-based services convergence and multi-clouds integration', *Computers in Industry*, vol. 64, no. 7, pp. 813–832, Sep. 2013.

[197] Q. Li, Z. Wang, W. Li, J. Li, C. Wang, and R. Du, 'Applications integration in a hybrid cloud computing environment: modelling and platform', *Enterprise Information Systems*, vol. 7, no. 3, pp. 237–271, 2013.

[198] Wikipedia, 'CRON (Unix).' [Online]. Available: http://es.wikipedia.org/wiki/Cron_%28Unix%29. [Accessed: 09-Jun-2015].

[199] A. Rico, ‘Globalgest ERP - Software de Gestión Cloud’, *desarrollotic.com*. [Online]. Available: http://globalgesterp.com/. [Accessed: 14-Jul-2013].

[200] A. Rico, ‘Desarrollo TIC. SEO, Web, and Software Development.’, *desarrollotic.com*. [Online]. Available: http://www.desarrollotic.com/. [Accessed: 23-Feb-2013].

[201] N. Niu, L. Da Xu, J. C. Cheng, and Z. Niu, ‘Analysis of Architecturally Significant Requirements for Enterprise Systems’, *IEEE Systems Journal*, vol. PP, no. 99, pp. 1–1, 2013.

[202] L. Da Xu, ‘Information architecture for supply chain quality management’, *International Journal of Production Research*, vol. 49, no. 1, pp. 183–198, 2011.

[203] F. Tao, Y. Cheng, L. Xu, L. Zhang, and B. Li, ‘CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System’, *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1435–1442, May 2014.