Universidad Carlos III de Madrid
Doctorado en Ciencia y Tecnología
Informática

Modelo de Interoperabilidad para Plataformas de Cloud Computing basado en Tecnologías del Conocimiento

AUTOR:

ENRIQUE JIMÉNEZ DOMINGO

TUTORES:

JUAN MIGUEL GÓMEZ BERBÍS

ISRAEL GONZÁLEZ CARRASCO

Leganés, Octubre 2013

TESIS DOCTORAL

Modelo de Interoperabilidad para Plataformas de Cloud Computing basado en Tecnologías del Conocimiento

Autor: **Enrique Jiménez Domingo**

Director/es: **Juan Miguel Gómez Berbís**
Israel González Carrasco

Firma del Tribunal Calificador:

Firma

Presidente:

Vocal:

Secretario:

Calificación:

Leganés, de de

Summary

In order to satisfy the requirements for the International doctorate mention, the summary of this Ph.D. thesis has been written in English. This summary includes methodology, main contributions and conclusions. Chapter 7 has also been written in English.

The Internet of services has altered the Web from a simple repository of information to become a services and transactions platform where organizations offer different kinds of services, increasing their business processes through the Web. With this transformation, new paradigm mergers have occurred, such as Cloud Computing and Software-as-a-Service, whose models are adapted to this new conception and promise the creation of new levels of efficiency through large scale sharing of functionalities and resources. With all of this, Cloud Computing related technologies have increased their presence and importance in the world of information technologies and have evolved to become a mature set of technological innovations able to provide a strong infrastructure for the SaaS paradigm. Cloud Computing and Software-as-a-Service open the doors to large scale economies and new horizons, but they have to face an important number of challenges, among which we can find the following:

- The lack of trusted models in order to determine the conditions under which it would be worthwhile for the organization to start a process of migration to these kinds of models.
- The lack of tested methods (for example, architecture guides) that would facilitate a potential migration.
- The lack of integration methods at different levels. This complicates the development and delivery processes of software that is able to communicate with other elements in the cloud platform, therefore causing to lose one of its most important features.

In dealing with this last aspect, it is possible to find the concept of interoperability. Interoperability has always been considered as one of the major challenges in information systems given that the heterogeneity of systems raises the complexity of the necessary methods to achieve it. This problem is growing with the increasing proliferation of independent-built information systems, which obstruct or impede the information exchange between them. Due to this, one of the most interesting properties of Cloud Computing systems is missing and, furthermore, it is restricting the progress towards the possibility of a completely interoperable world where the access to information would be nearly infinite.

The problem of interoperability has been faced from different perspectives, however the ones based on semantic technologies, where reaching interoperability is historically one of their major goals, are especially interesting. The Semantic Web was conceived for in order to solve the problems of data overloading and heterogeneity in the Web, which causes the absence of interoperability among other features. The approach of Semantic Web is to add semantic to the data enabling the machine´s capability to process, reason, combine and make logical deductions with them in a similar way than a human will do, providing a set of useful tools.

Within this field, this doctoral thesis brings together the current most relevant and rising concepts like Cloud Computing, SaaS, semantic technologies, business process modeling, systems interoperability, etc. The goal is to encourage the evolution of new platforms oriented towards interoperability and cost reduction, which can have a significant impact in the industry. The research accomplished takes these technologies and paradigms as a base for the definition of a standard or common language that allows the interoperability between applications hosted in the same Cloud Computing environment. With this, it is possible to take a big step forward in the technology areas previously mentioned, creating a clearly innovative approach with results that will obtain great significance in the world of information systems.

With the information mentioned, the solution framed within this doctoral thesis is oriented towards the design of a semantic language ontology-based and of a global system that works around it, able to capture the knowledge of the applications hosted in a Cloud Computing structure, setting up the development of information exchange processes between heterogeneous applications in an automatic manner. Therefore, this thesis proposes the combination of Cloud Computing and Semantics in order to tackle the problem of interoperability at application level. Currently this approach is not widespread due to the "recent" explosion of cloud systems, but the benefits that they can mutually provide translate into a great motivation for working and deeply researching in this field.

The research methodology followed during this doctoral thesis in order to achieve the goals and provide the formulated hypothesis consists of:

- State of the art study. Through the study of related works, it is possible to know the methods and technologies necessary to achieve the outlined goals. This study is divided in five main categories:
    - Cloud Computing.
    - Semantic Web.
    - Web Services.
    - Semantic Web Services.
    - Interoperability.

- Analysis of the Literature's most representative aspects. The analysis of the different concepts addressed in the state of the art, allows performing a correct and strongly supported decision process to utilize the most appropriate technologies and tools for the problem being solved.

- First approach of the solution. This stage allows for the layout of the foundations and requirements that have to be fulfilled in order to reach the proposed objectives. This researching phase makes use of the conclusions extracted from the previous analysis and allows to determine the viability of the researching as well as checking whether the selected tools are good enough to achieve the desired solution.

- Final design of the solution. In this phase the design process concludes. The whole set of requirements have to be fulfilled and it must be detailed enough to be able to start the implementation of the environment for its validation.

- Language implementation and configuration of Cloud Computing platform. This stage indicates the beginning of the validation process which will allow the testing of whether the design meets the objectives and is aligned with the proposed hypothesis. The configuration of the Cloud Computing platform will enable evaluating and validating the researching in a real environment.

- Validation. The whole designed and developed functionality is tested within this phase in order to ensure that the problem is correctly represented and the system is able to perform the different tasks to accomplish automatic interoperability.

- Hypothesis evaluation and analysis of results. After validating the proposed hypothesis, results are analyzed to extract the necessary conclusions related to the implemented research.

- Documentation. This task has been developed throughout the whole process of this doctoral thesis preparation. In the first stages, the documentation registers the results obtained from the state of the art study and the analysis of the problem. During the following phases, all the features of the developed design and the relevant aspects from the global system were documented. Within the validation phase, the results of the performed procedures have been written. Lastly, the definitive statement that makes this document was composed, including the conclusions extracted from each of the aspects relative to this doctoral thesis, arising from the documentation generated along the different phases.

The evaluation of the language, and the global system where it is framed, has proved the viability of the proposed solution, achieving automatic interoperability of heterogeneous applications at data level in Cloud Computing environments. Thanks to the joint action of all components forming the designed architecture, this research provided satisfactory results in the information exchange processes between applications hosted in the same Cloud Computing infrastructure. Moreover, the validation process has allowed for the demonstration of each of the hypotheses proposed by the research. By the information contained herein, it is possible to affirm that all the objectives exposed in this doctoral thesis have been accomplished, resulting in a great success and a relevant contribution to the different areas analyzed. Furthermore, the conclusions extracted from the whole research allow for the extrapolation of some theoretical aspects to other environments, as well as open new research lines that contribute to the achievement of interoperable environments that are able to provide a better and more efficient use of the vast volume of existing information.

Resumen

A fin de cumplir los requisitos necesarios para la obtención de la mención internacional del doctorado, el resumen de esta tesis fue originalmente escrito en inglés. A continuación se incluye la versión en lengua española.

La Internet de los servicios ha ido modificando la Web desde un mero repositorio de información hasta convertirse en una plataforma para servicios y transacciones donde las organizaciones ofertan servicios de todo tipo, incrementando sus procesos de negocio exponiéndolos a través de la Web. Unido a esta transformación surgen nuevos paradigmas, como el Software-as-a-Service y el Cloud Computing, cuyos modelos se adaptan a esta nueva concepción y prometen crear nuevos niveles de eficiencia mediante la compartición de funcionalidades y recursos a gran escala. Con todo ello, las tecnologías relacionadas con el Cloud Computing han ido incrementando su presencia e importancia en el mundo de las tecnologías de la información y han evolucionado hasta convertirse en un conjunto maduro de innovaciones tecnológicas capaces de proporcionar una sólida infraestructura para el paradigma SaaS. El Cloud Computing y el SaaS abren las puertas a economías de gran escala y también a nuevos horizontes pero se encuentran con un importante número de retos que afrontar entre los que se encuentran:

- La falta de modelos contrastados para determinar bajo qué condiciones es rentable para las organizaciones el proceso de migración hacia estos modelos.
- La falta de métodos probados (por ejemplo, guías de arquitecturas) para facilitar dicha migración.
- La falta de métodos de integración a diferentes niveles, hecho que dificulta el proceso de desarrollo y entrega de un software capaz de comunicarse con otros elementos de la plataforma cloud, perdiendo una de sus características más importantes.

Relacionado con el último aspecto comentado se encuentra el concepto de la interoperabilidad. La interoperabilidad siempre se ha considerado como una de las asignaturas pendientes de los sistemas de información, ya que la heterogeneidad de los sistemas eleva la complejidad de los medios necesarios para alcanzarla. Este problema se está acrecentando con la creciente proliferación de sistemas de información construidos de forma independiente que dificultan o impiden los intercambios de información entre ellos. Debido a esto se pierde una de las propiedades más interesantes de los sistemas de computación en nube pero además se limita el avance hacia la posibilidad de un mundo completamente interoperable donde el acceso a la información sería prácticamente infinito.

El problema de la interoperabilidad se ha afrontado desde diversas perspectivas, pero resultan especialmente interesantes los enfoques realizados por las tecnologías semánticas, donde alcanzar la interoperabilidad a distintos niveles es históricamente uno de sus objetivos principales. La Web Semántica fue concebida principalmente para solucionar los problemas de sobrecarga y heterogeneidad de los datos en la Web, hecho que trae como consecuencia la ausencia de interoperabilidad. Su enfoque es el de añadir semántica a los datos de manera que las máquinas sean capaces de procesarlos, razonar con ellos, combinarlos y realizar

deducciones lógicas de manera muy similar a como lo haría un ser humano, proporcionando para ello una serie de herramientas útiles.

Dentro de este ámbito, esta tesis doctoral reúne los conceptos más relevantes y emergentes en la actualidad como Cloud Computing, SaaS, tecnologías semánticas, modelado de procesos de negocio, interoperabilidad entre sistemas, etc. El objetivo es fomentar la evolución de nuevas plataformas orientadas en torno a la interoperabilidad y la reducción de costes que puede suponer un impacto significativo en la industria. La investigación realizada, toma estas tecnologías y paradigmas como base para poder definir un estándar o lenguaje común que permita la interoperabilidad entre aplicaciones dentro de un entorno de Cloud Computing. Con esto se consigue dar un paso importante en las áreas de las tecnologías mencionadas, suponiendo un enfoque claramente innovador y cuyos resultados resultarán de gran relevancia dentro del ámbito de los sistemas de información.

Por todo ello, la solución enmarcada dentro de esta tesis doctoral está orientada al diseño de un lenguaje semántico basado en ontologías y de un sistema global que trabaja a su alrededor capaz de capturar el conocimiento de las aplicaciones alojadas en una estructura de Cloud Computing habilitando la realización de procesos de intercambio de información entre aplicaciones heterogéneas de forma automática. Esta tesis doctoral plantea, por tanto, la combinación del Cloud Computing y la Semántica para afrontar el problema de la interoperabilidad a nivel de aplicación. Este enfoque no se encuentra extendido en la actualidad debido a la "reciente" explosión de los sistemas cloud, pero los beneficios que pueden aportarse mutuamente suponen una gran motivación para trabajar e investigar en profundidad sobre ello.

La metodología de investigación seguida en esta tesis doctoral para alcanzar los objetivos y demostrar las hipótesis planteadas ha consistido en:

- Estudio del estado de arte. Mediante el estudio de los trabajos relacionados se pueden conocer los métodos y tecnologías necesarios para alcanzar los objetivos planteados. Este estudio se ha dividido en cinco bloques principales:
    - Cloud Computing.
    - Web Semántica.
    - Servicios Web.
    - Servicios Web Semánticos.
    - Interoperabilidad.

- Análisis de los aspectos más representativos de la literatura. El análisis de los diferentes conceptos tratados en el estado del arte, permite realizar una toma de decisiones correcta y sólidamente fundamentada para utilizar las tecnologías y herramientas más adecuadas al problema.

- Presentación de una primera aproximación de la solución. Esta etapa permitirá sentar las bases y requisitos que se deben cumplir para cubrir los objetivos propuestos. Esta parte de la investigación emplea las conclusiones extraídas del análisis previo y

permitirá concluir la viabilidad de la investigación y también si las herramientas seleccionadas son suficientes para alcanzar la solución deseada.

- Diseño final de la solución. En esta fase se concluye el proceso de diseño. Todos los requisitos han de quedar cubiertos y debe presentar un nivel de detalle suficiente para poder comenzar la implementación del entorno para su validación.

- Implementación del lenguaje y configuración de la plataforma de Cloud Computing. Esta fase marca el inicio de la validación que permitirá comprobar que el diseño cumple con los objetivos marcados y que se encuentra en la línea de las hipótesis propuestas. La configuración de la plataforma de computación en nube permitirá evaluar y validar la investigación en un entorno real.

- Validación. En esta fase se comprueba que toda la funcionalidad diseñada y desarrollada permite representar el conocimiento relativo al problema y la interoperabilidad de las aplicaciones en la nube sin necesidad de intervención por parte del usuario.

- Evaluación de hipótesis y análisis de resultados. Tras validar las hipótesis propuestas, se analizan los resultados para extraer las consiguientes conclusiones de toda la investigación realizada.

- Documentación. La tarea de documentación se ha llevado a cabo a lo largo de todo el proceso de elaboración de la presente tesis doctoral. En las primeras fases, la tarea de documentación registra los resultados del estudio del estado del arte y del análisis del problema. Durante el desarrollo de las siguientes etapas, se documentaron las características del diseño realizado y los aspectos relevantes del sistema global. En la fase de validación, se han documentado los resultados de los procedimientos de validación seguidos. Finalmente se redactó la documentación definitiva que constituye la presente memoria, incluyendo las conclusiones extraídas de cada uno de los aspectos relativos a la tesis doctoral, a partir de la documentación generada a lo largo de las distintas fases.

La evaluación del lenguaje y del sistema global en el que se enmarca, ha probado la viabilidad de la solución presentada para alcanzar la interoperabilidad entre aplicaciones heterogéneas a nivel de datos en entornos de computación en nube de manera automática. Gracias a la acción conjunta de todos los componentes integrantes de la arquitectura diseñada se han conseguido resultados satisfactorios en procesos de intercambio de información entre aplicaciones situadas en una misma estructura de Cloud Computing. El proceso de validación ha permitido además demostrar cada una de las hipótesis planteadas por la investigación. Por todo ello es posible afirmar que se han alcanzado los objetivos planteados por la tesis doctoral, resultando finalmente en un gran éxito y una contribución de relevancia en los diferentes ámbitos estudiados. Además, las conclusiones extraídas de la totalidad de la investigación permiten extrapolar aspectos teóricos a otros medios y abrir nuevas líneas de investigación destinadas a contribuir a la consecución de entornos interoperables que permitan un mejor y más eficaz aprovechamiento de la información existente.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1 Introducción

El propósito de este capítulo es realizar una breve descripción del problema tratado y situar el contexto y la motivación que han servido de base para la investigación realizada. La interoperabilidad es un problema que ha estado presente desde hace décadas y cuya solución se ha tratado desde distintas perspectivas sin alcanzarse una solución aplicable a todos los escenarios. La aparición del Cloud Computing como paradigma emergente que ha cobrado gran importancia en muy poco tiempo ha hecho que, además de las ventajas propias de estos sistemas, se presenten como entornos adecuados para profundizar y sacar el máximo provecho a sus propiedades. La investigación presentada en esta tesis doctoral reúne varias tecnologías y enfoques punteros en un único trabajo de gran carácter innovador que soluciona el problema de la interoperabilidad desde un punto de vista original e innovador.

Una vez explicada la motivación que ha llevado a la realización de este trabajo y definidos los objetivos perseguidos, se justifica la necesidad de este trabajo. Finalmente se detalla la metodología diseñada para la consecución de los objetivos previamente enunciados y se realiza un breve resumen de la estructura del documento presentado.

1.1 Contexto

La Internet de los servicios está modificando la Web transformándola en una plataforma para servicios de negocio y transacciones basadas en el intercambio de información. Durante los últimos años, la tendencia para las grandes organizaciones ha sido la de incrementar sus procesos de negocio exponiéndolos a través de la Web para grandes desarrollos de software, así como la compartición de sus servicios tanto dentro como fuera de la propia organización. A pesar de esto y de manera reciente, han surgido nuevos paradigmas para la Ingeniería del Software y de servicios, como pueden ser el Software-as-a-Service (SaaS) o el Cloud Computing (o computación en nube), cuyos modelos prometen crear nuevos niveles de eficiencia mediante la compartición de funcionalidades y recursos computacionales a gran escala. SaaS es un modelo de distribución de software en el que además de proporcionar ese software, el suministrador ofrece servicios adicionales como mantenimiento y soporte. La ventaja de SaaS reside en que el software es distribuido (centralizado) y alojado en la red, eliminando el requisito para los usuarios de instalar el software en sus propias infraestructuras, así como ningún tipo de dato relacionado.

Estrechamente ligado con el modelo SaaS aparece el Cloud Computing. Este paradigma se define como un modelo de infraestructura de aprovisionamiento de las tecnologías de la

información en el que las aplicaciones y los servicios provenientes de varias organizaciones se encuentran alojadas en una única infraestructura física también ofrecida como servicio. El Cloud Computing y el SaaS abren las puertas a economías de gran escala y también a nuevos horizonte pero se encuentran con un importante número de retos que afrontar. Algunos de los más relevantes son los siguientes:

- La falta de modelos contrastados para determinar bajo qué condiciones es rentable para las organizaciones el proceso de migración hacia estos modelos, teniendo en cuenta factores tanto legales como técnicos y de negocios.
- La falta de métodos probados (por ejemplo, guías de arquitecturas) para facilitar dicha migración
- La falta de métodos de integración a diferentes niveles, hecho que dificulta el proceso de desarrollo y entrega de un software capaz de comunicarse con otros elementos de la plataforma cloud, perdiendo una de sus características más importantes.

Otros retos específicos incluyen cómo representar las capacidades y requisitos de los sistemas SaaS y Cloud Computing y cómo permitir a los brokers realizar el proceso de unión entre dichas capacidades y requisitos.

Con todo ello, las tecnologías relacionadas con el Cloud Computing han ido incrementando su presencia e importancia en el mundo de las tecnologías de la información y han evolucionado hasta convertirse en un conjunto maduro de innovaciones tecnológicas capaces de proporcionar una sólida infraestructura para el paradigma SaaS. Aún así, todavía existen numerosas preguntas acerca del desarrollo de aplicaciones enfocadas a la nube así como de su interoperabilidad (no conseguida hasta el momento) y su integración, surgidas de todas las ramas de la Informática y también en el ámbito del Platform-as-a-Service (PaaS).

En relación con esto último, se puede definir la interoperabilidad como la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada (Geraci, 1991). La interoperabilidad siempre se ha considerado como una de las asignaturas pendientes de los sistemas de información, ya que la heterogeneidad de los mismos eleva la complejidad de los sistemas necesarios para alcanzarla. Esto hace que se pierda parte del potencial de las propias aplicaciones y servicios, que verían multiplicadas sus prestaciones si pudiesen compartir y utilizar los datos proporcionados por otros sistemas. Este problema se ha afrontado desde diversas perspectivas, pero resultan especialmente interesantes los enfoques realizados por las tecnologías semánticas, donde alcanzar la interoperabilidad a distintos niveles es uno de los objetivos principales. La Web Semántica fue concebida principalmente para solucionar los problemas de sobrecarga y heterogeneidad de los datos en la Web, hecho que trae como consecuencia la ausencia de interoperabilidad. Su enfoque es el de añadir semántica a los datos de manera que las máquinas sean capaces de procesarlos, razonar con ellos, combinarlos y realizar deducciones lógicas de manera muy similar a como lo haría un ser humano. Para ello, proporcionan una serie de herramientas que pueden ser utilizadas para alcanzar estos objetivos.

Esta tesis doctoral reunirá varios de los conceptos más relevantes y emergentes en la actualidad como Cloud Computing, SaaS, tecnologías semánticas, modelado de procesos de negocio o la interoperabilidad entre sistemas, etc., para fomentar la evolución de nuevas

plataformas orientadas en torno a la interoperabilidad y la reducción de costes que puede suponer un impacto significativo en la industria. La investigación realizada, toma estas tecnologías y paradigmas como base para poder definir un estándar o lenguaje común que permita la interoperabilidad entre aplicaciones dentro de un entorno de Cloud Computing. Con esto se consigue dar un paso importante en el ámbito de las tecnologías mencionadas, suponiendo un enfoque claramente innovador y cuyos resultados resultarán de gran relevancia dentro del ámbito de los sistemas de información.

1.2 Motivación

La realización de esta tesis doctoral viene motivada por el gran interés investigador que tendría tanto para la comunidad científica en general como para la semántica y las nuevas tecnologías en particular. Además de esto, supone también un gran efecto incentivador por las posibilidades que podría ofrecer a nivel empresarial y comercial, ya que esta investigación realiza aportaciones relevantes para facilitar la comunicación entre entidades heterogéneas y solucionando algunos de los grandes problemas que existen en este campo en la actualidad.

El paradigma del Cloud Computing ha cobrado una gran relevancia en los últimos años y hoy aparece como una solución adecuada en diversos ámbitos. Se ha posicionado con fuerza en mercados diversos y sus características le hacen muy atractivo para empresas de diferentes tamaños y sectores. Son algunas de esas características las que lo sitúan como un marco perfecto para la investigación desarrollada.

Los sistemas cloud permiten crear entornos “cerrados” en los que definir una serie de restricciones, limitando los conflictos que se suceden en ecosistemas heterogéneos. Estos requisitos ayudarán a conseguir la información necesaria para establecer un canal de comunicación. Por ello, se hace difícil encontrar un ambiente más adecuado que el que proporciona el Cloud Computing para el desarrollo de la investigación.

Las tecnologías semánticas nacieron con el objetivo de comprender mejor los contenidos existentes en la Web y en los sistemas de información, así como para poder interconectar sistemas diversos para facilitar el intercambio de información. Estas técnicas han afrontado el problema de diferentes formas, empleando todas las herramientas y recursos a su alcance, pero la diversidad y cantidad de los datos a menudo han frenado los resultados en este ámbito. Sin embargo, sus propiedades hacen que su aplicación sea necesaria para alcanzar los objetivos propuestos.

De todo esto nace una de las principales motivaciones para la realización de esta tesis doctoral y que supone un avance adicional dentro de este ámbito: la inexistencia de un lenguaje común o estándar que permita la compartición de información de forma sencilla. La subsanación de esta carencia supone una gran motivación y a la vez un importante reto que se afronta con esta investigación.

En esta tesis doctoral se plantea la combinación del Cloud Computing y la Semántica para afrontar el problema de la interoperabilidad a nivel de aplicación. Este enfoque no se

encuentra extendido en la actualidad debido a la "reciente" explosión de los sistemas cloud, pero los beneficios que pueden aportarse mutuamente hacen que sea una motivación más que suficiente para trabajar e investigar en profundidad sobre ello. Los resultados obtenidos por este trabajo arrojarán luz sobre este problema y seguro aportarán información relevante para futuras investigaciones en este ámbito.

De manera paralela a la tesis doctoral realizada y debido al interés principal de la misma, se ha participado activamente en el crecimiento de dos proyectos de investigación de convocatorias competitivas. El primero de ellos perteneciente titulado "*SITIO: Semantic Business Processes based on Software-as-a-Service and Cloud Computing"*, financiado por el Ministerio de Industria, Turismo y Comercio dentro del PLAN AVANZA, que además obtuvo el sello de calidad europeo EUREKA. El segundo proyecto, llamado "FLORA: Financial Linked Open Data Reasoning and Management for Web Science", financiado por CICYT. A pesar de que ambos proyectos poseían unos objetivos diferentes a los que se plantean en esta investigación, poseen líneas de investigación relacionadas con las que son objeto de estudio de esta tesis doctoral y los avances logrados contribuyeron de manera notable a aumentar la motivación de seguir trabajando en estas áreas.

1.3 Descripción del problema

Esta tesis doctoral se centra en la investigación de uno de los mayores problemas existentes en los sistemas de la información: la interoperabilidad. Si bien éste no es el único problema que se pretende resolver, sí que puede ser considerado como el principal y más relevante para la comunidad investigadora. Como se comentaba anteriormente, el IEEE define la interoperabilidad como la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada (Geraci, 1991). A esta definición habría que añadir que la dificultad de conseguir esta comunicación se ve incrementada cuando se trata de sistemas heterogéneos, construidos de manera independiente y en base a reglas, formatos y lenguajes diferentes. Más próxima a esta definición se encuentra la proporcionada por el Marco Iberoamericano de Interoperabilidad ("MarcoIberoamericano," 2010), la cual se encuentra muy en línea con la dada por la Comisión Europea, en la que se define la interoperabilidad como la habilidad de organizaciones y sistemas dispares y diversos para interaccionar con objetivos consensuados y comunes con la finalidad de obtener beneficios mutuos. Ésta es la situación habitual de los sistemas de información en la actualidad. Cada uno de ellos se encuentra diseñado para cumplir un propósito sin tener en cuenta aspectos de interoperabilidad o comunicación con otros sistemas, de forma que su intercambio de información se complica enormemente.

El principal problema derivado de todo esto es la falta de un estándar o lenguaje común que permita a diversas aplicaciones compartir información de forma sencilla. Al propio problema técnico que conlleva el alcanzar este objetivo, se unen aspectos de negocio. Hoy en día parece un imposible que empresas y desarrolladores lleguen a un acuerdo por el que se apruebe un sistema común de compartición de datos. Cada empresa tiene sus sistemas y proyectos y no desean cambiar sus métodos de trabajo. Los grandes organismos intentan imponer sus bases,

pero ni siquiera entre ellos ceden y tratan de imponerse unos a otros, con lo que las estrategias de negocio implícitas no tienen perspectiva de cambiar.

Por todo ello, no existe en la actualidad ningún sistema capaz de afrontar esta situación, lo que otorga a la investigación un valor añadido al tratar de responder a todas las preguntas que rodean a este problema desde un punto de vista novedoso, innovador y multidisciplinar.

1.4 Objetivos

1.4.1 Objetivo General

El objetivo central esta tesis doctoral será la creación de un estándar o lenguaje común orientado a la interconexión o interoperabilidad de aplicaciones de diversa índole en un entorno de Cloud Computing. Este lenguaje servirá de base o sustento para las aplicaciones que deseen interactuar entré sí en el interior de la nube, estableciéndose de esta forma unos mínimos prerrequisitos que permitan asegurar el correcto intercambio de información entre dichas aplicaciones. Además se añadirá semántica al lenguaje de forma que permita la descripción y etiquetación de las aplicaciones para posteriores búsquedas o procesos que precisen de información adicional, suponiendo un valor añadido en cuanto a aspectos de innovación se refiere. Mediante esta definición se soluciona el problema de la complejidad que supone establecer un modelo diferente para cada aplicación, algo que sería completamente inviable. De esta manera servirá de forma general a todos aquellos módulos que se encuentren o se añadan a la infraestructura de la nube.

1.4.2 Objetivos Específicos

De forma desglosada, los objetivos individuales que se pretenden obtener en esta investigación son los siguientes:

1. Realizar un estudio detallado del estado del arte en las áreas de influencia de la investigación que permitan un conocimiento profundo de los trabajos realizados y las carencias existentes para abordar el problema de forma directa.
2. Definir un modelo de intercambio de información general o lenguaje que permita representar el conocimiento contenido en las aplicaciones existentes en las plataformas de computación en nube.
3. Añadir semántica al lenguaje definido para poder utilizar el conocimiento capturado para realizar inferencia y operaciones lógicas que incrementen las posibilidades del modelo.
4. Dar solución a las interacciones y transacciones que se produzcan dentro de aplicaciones que se encuentren en una misma estructura de Cloud Computing.

5. Permitir el intercambio de datos de manera sencilla sin necesidad de intervención adicional por parte del usuario (automatización). Esto supone una mejora muy relevante a nivel colectivo, facilitando además la reutilización y la suscripción de usuarios a aplicaciones diversas.
6. Proporcionar una solución unificada en todos los aspectos de la empresa (tanto desde el punto de vista de la propia empresa y su gestión interna como del usuario que quiere acceder o ser cliente de ella) contando únicamente con un ordenador con conexión a Internet, de manera que permita el acceso a la plataforma que posee todas las aplicaciones necesarias.
7. Diseñar un sistema que sirva para validar la solución propuesta en esta investigación y permita comprobar el potencial de los aspectos planteados y el aumento cualitativo que supone la consecución de los objetivos en cuanto a nivel de prestaciones y usabilidad de la plataforma a la que se apliquen.

1.5 Justificación

La interoperabilidad entre sistemas ha sido una de las metas que ha perseguido la Informática desde sus inicios (Park & Ram, 2004). La problemática que supone establecer una comunicación entre sistemas heterogéneos ha generado diversas aproximaciones y enfoques (Hervé Panetto & Molina, 2008) (Morris, Levine, Meyers, Place, & Plakosh, 2004) (Zhao, Deng, & Shen, 2001), pero en ninguno de ellos se ha logrado un consenso que haya permitido diseñar soluciones adecuadas. En muchos casos se ha optado por realizar configuraciones uno a uno, pero en la gran mayoría de los entornos esta opción resulta inviable.

Conseguir sistemas interoperables supone dar un paso de gigante hacia una red de información que se podría considerar prácticamente infinita (A. P. Sheth, 1999). Cada uno de los sistemas sería capaz de, sin intervención alguna del usuario, conseguir la información que se necesite de forma rápida y eficiente mediante la comunicación directa con los sistemas de los que precise y que contienen los datos deseados. A pesar de que esto se plantea como una situación ideal que en la actualidad resulta difícil de imaginar, esta investigación pretende acercarse a la idea de un mundo completamente interoperable.

Las dificultades a afrontar son muchas y variadas (Kasunic, 2001), pero la investigación realizada en esta tesis doctoral pretende presentar un modelo de conocimiento o lenguaje basado en tecnologías y paradigmas punteros que proporcione una solución válida al problema de la interoperabilidad. La investigación aprovecha las ventajas de cada una de estas tecnologías para que esta conjunción de paradigmas no estudiada con anterioridad en la literatura, proporcione una solución eficaz, original e innovadora.

1.5.1 ¿Por qué es importante el estudio de la interoperabilidad?

La interoperabilidad es una propiedad de la que hoy en día carecen una gran mayoría de los sistemas de información (Sciore, Siegel, & Rosenthal, 1994). Han existido varias iniciativas por parte de organizaciones y compañías para crear estándares de interoperabilidad o marcos que regulasen estos aspectos (Chen & Doumeingts, 2003) (Berre et al., 2007). Sin embargo ninguno ha trascendido lo suficiente como para imponerse como un estándar generalmente aceptado. Por ello, el estudio realizado en este apartado deja claro que la interoperabilidad es un tema que despierta gran interés y que requiere estudios e investigaciones profundas (Guijarro, 2007).

Para comprender la complejidad, el éxito y el fracaso de estos intentos es necesario entender que los sistemas ya están desarrollados, existen luchas de poder, estrategias comerciales y empresariales. Estos y otros muchos factores influyen a la hora de alcanzar acuerdos para establecer consensos en torno a estos asuntos. Ninguna entidad, organización o gobierno desea ceder o cambiar sus sistemas en favor de otros, en gran parte por motivos de competencia y por la consiguiente inversión económica que conlleva un cambio en la estrategia de negocio (Lagoze & Van de Sompel, 2001).

Sin embargo, si existiese algún medio por el que sistemas basados en tecnologías diferentes, provenientes de ámbitos dispares y con objetivos y tamaños distintos fuesen capaces de intercambiar información sin necesidad de modificar sus estructuras, muchos de estos problemas quedarían resueltos. De esta manera se abriría un nuevo horizonte para unificar contenidos que en último término puede llevar a un cambio en las estrategias empresariales.

1.5.2 ¿Por qué centrarse en entornos de Cloud Computing?

Los sistemas de información se encuentran presentes en prácticamente la totalidad de las actividades profesionales de la actualidad. En pocos años han llegado a ser una parte básica e insustituible de cualquier tipo de negocio por su capacidad de procesamiento y almacenamiento (Pawlak, 1981). Muchos de ellos, y el número sigue creciendo, aportan además conocimiento adicional, permitiendo a las entidades un mejor posicionamiento en sus mercados y un mejor aprovechamiento de las capacidades de su empresa. Estos hechos son objetivos y observables, pero de todo ello se puede sacar una importante conclusión: la cantidad de sistemas y los datos que contienen están en continuo crecimiento y se hacen cada vez más difíciles de gestionar y manejar, o lo que es lo mismo, el mundo es demasiado grande para gestionarse de manera global.

Debido a esto, es necesario encontrar un ámbito en el que se pueda acotar el problema y hacerlo manejable para un estudio detallado. Es en este punto donde aparece el paradigma del Cloud Computing. La computación en nube ha irrumpido con fuerza debido a sus ideas de software distribuido, pago por uso, mantenimiento personalizado, almacenamiento "ilimitado"… Sus características lo han hecho ideal para adaptarse a las necesidades de pequeñas empresas que ven cómo pueden reducir su inversión inicial, a los dispositivos

móviles con poca capacidad de almacenamiento que acceden a sus contenidos a través de la nube y a grandes organizaciones que además de ver reducidos sus gastos, observan nuevas oportunidades para ofrecer sus servicios (R. Buyya, Yeo, & Venugopal, 2008). Estas propiedades hacen del Cloud Computing un entorno más que adecuado para presentar servicios.

En esta investigación además de aprovechar todas las ventajas que aporta el paradigma, se utilizará la infraestructura cloud como un entorno cerrado o "jardín vallado" (término que se definirá en mayor profundidad en la sección 4.2) en el que se podrán fijar una serie de requisitos para las aplicaciones que quieran alojarse en esa infraestructura de nube. De esta manera, al cumplir estos requisitos, se conoce cierta información de las aplicaciones existentes en la nube y se limita la cantidad de datos e información a manejar, permitiendo solucionar las incompatibilidades que se puedan producir. Así, además de investigar en una de las tendencias punteras actualmente, se podrá limitar el ámbito para hacer más manejable el problema y realizar una validación adecuada que permita obtener resultados de forma general.

1.5.3 ¿Por qué se plantea un enfoque basado en lenguajes y Web Semántica?

El gran éxito de la Web reside, entre otras cosas, en la facilidad de acceder a una enorme cantidad de contenidos en continuo crecimiento y a su capacidad para establecer comunicaciones de todo tipo a bajo coste. Esta continua evolución ha provocado un estancamiento a la hora de utilizar la información que contiene. Cada vez se hace más complicado encontrar con exactitud aquello que se está buscando. Esta sobrecarga y heterogeneidad de fuentes hace que se incremente el problema de la interoperabilidad (T. Berners-Lee, Hendler, & Lassila, 2001).

Con el objetivo de solucionar estos problemas nace la Web Semántica (T. Berners-Lee et al., 2001). Añadiendo significado a los datos se consigue que las máquinas puedan tener cierto conocimiento de ellos y de esta manera realizar procesos lógicos de manera similar a como lo harían los seres humanos.

Centrado en ese objetivo, la Web Semántica proporciona una serie de herramientas que permiten añadir semántica a los datos para poder razonar con ellos. Entre estas herramientas destacan las ontologías (Dieter Fensel, 2001). Las ontologías describen el conocimiento de un dominio y su compartición hace que distintas aplicaciones puedan compartir la misma información (Chandrasekaran, Josephson, & Benjamins, 1999). Partiendo desde este principio, la Web Semántica y en particular las ontologías, permitirán alinear sus objetivos con el objetivo de la interoperabilidad perseguido por la investigación de esta tesis doctoral. De este modo se diseñará un lenguaje basado en tecnologías semánticas que capture el conocimiento de las aplicaciones contenidas en un entorno de computación en nube y que sea capaz de solucionar las incompatibilidades surgidas de sistemas heterogéneos mediante inferencia y operaciones lógicas.

1.5.4 ¿Qué aportaciones tendrá esta tesis doctoral?

La investigación realizada permitirá la creación de una estructura semántica interoperable de computación en nube, en las que las aplicaciones que se encuentren alojadas en ella podrán intercambiar información de manera sencilla y sin intervención por parte del usuario.

A lo largo del documento se mostrarán las diferentes capacidades de las tecnologías empleadas y su adecuación con el problema a resolver. Todas ellas son seleccionadas tras un estudio detallado de sus propiedades mediante el análisis de la literatura y son reunidas para conseguir el enfoque único con requiere esta investigación. Con ello se demostrará que estas tecnologías son apropiadas para enfrentar el problema del que se ocupa la tesis doctoral y que se pueden realizar investigaciones alternativas dentro de la misma línea proporcionada en este documento.

Asimismo el desarrollo de una plataforma de Cloud Computing permitirá validar los resultados obtenidos por la investigación. Se evaluarán y validarán los diferentes aspectos del lenguaje que tengan incidencia con su funcionalidad y se demostrará que se consigue interoperabilidad entre aplicaciones heterogéneas.

1.6 Metodología de la investigación

Este apartado detalla las actividades en las que se ha dividido la investigación para alcanzar los objetivos expuestos anteriormente. Para la resolución del problema en cuestión es necesario realizar una serie de pasos clave que se detallan a continuación (Figura 1):

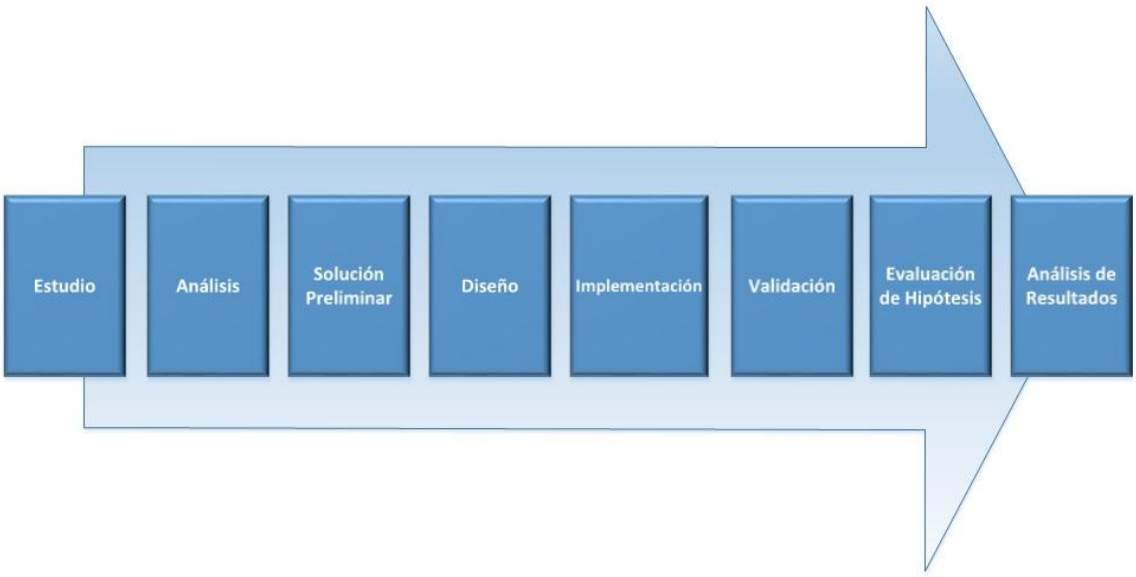


Figura 1. Etapas de la investigación

- Estudio del estado de arte. Mediante el estudio de los trabajos relacionados se pueden conocer los métodos y tecnologías necesarios para alcanzar los objetivos planteados. Este estudio se ha dividido en cinco bloques:
    - Cloud Computing
    - Web Semántica
    - Servicios Web
    - Servicios Web Semánticos
    - Interoperabilidad

- Análisis de los aspectos más representativos de la literatura. El análisis de los diferentes conceptos tratados en el estado del arte, permite realizar una toma de decisiones correcta y sólidamente fundamentada para utilizar las tecnologías y herramientas más adecuadas al problema.

- Presentación de una primera aproximación de la solución. Esta etapa permitirá sentar las bases y requisitos que se deben cumplir para cubrir los objetivos propuestos. Esta parte de la investigación emplea las conclusiones extraídas del análisis previo y permitirá concluir la viabilidad de la investigación y también si las herramientas seleccionadas son suficientes para alcanzar la solución deseada.

- Diseño final de la solución. En esta fase se concluye el proceso de diseño. Todos los requisitos han de quedar cubiertos y debe presentar un nivel de detalle suficiente para poder comenzar la implementación del entorno para su validación.

- Implementación del lenguaje y configuración de la plataforma de Cloud Computing. Esta fase marca el inicio de la validación que permitirá comprobar que el diseño cumple con los objetivos marcados y que se encuentra en la línea de las hipótesis propuestas. La configuración de la plataforma de computación en nube permitirá evaluar y validar la investigación en un entorno real.

- Validación. En esta fase se comprueba que toda la funcionalidad diseñada y desarrollada permite representar el conocimiento relativo al problema y la interoperabilidad de las aplicaciones en la nube sin necesidad de intervención por parte del usuario.

- Evaluación de hipótesis y análisis de resultados. Tras validar las hipótesis propuestas, se analizan los resultados para extraer las consiguientes conclusiones de toda la investigación realizada.

- Documentación. La tarea de documentación se ha llevado a cabo a lo largo de todo el proceso de elaboración de la presente tesis doctoral. En las primeras fases, la tarea de documentación registra los resultados del estudio del estado del arte y del análisis del problema. Durante el desarrollo de las siguientes etapas, se documentaron las características del diseño realizado y los aspectos relevantes del sistema global. En la fase de validación, se han documentado los resultados de los procedimientos de validación seguidos. Finalmente se redactó la documentación definitiva, que

constituye la presente memoria, a partir de la documentación generada a lo largo de las distintas fases.

1.7 Estructura de la Tesis Doctoral

En adelante, el presente documento se estructura de la siguiente manera:

- Estado del arte. En este capítulo se revisan los conceptos más importantes tratados en la investigación así como los trabajos de mayor relevancia realizados hasta la fecha en cada una de las áreas que abarca el trabajo.

- Propuesta de solución. Una vez definidos los problemas a solucionar con el desarrollo de esta tesis doctoral, este capítulo detalla las hipótesis que guían la investigación y se describe la aproximación de la solución propuesta.

- Diseño de la solución. A continuación se presentan las diferentes fases de diseño que se han desarrollado hasta alcanzar el diseño final tras analizar los trabajos propuestos y las tecnologías a emplear.

- Detalles de implementación. En este capítulo se comentan los aspectos más relevantes de la implementación de las diferentes partes que se han desarrollado para permitir la validación de la investigación realizada.

- Evaluación y Validación. Posteriormente se aborda la evaluación y validación del lenguaje propuesto y su aplicación en un dominio real. También se analizan los resultados obtenidos con el fin de extraer conclusiones relevantes sobre la investigación.

- Conclusiones y Líneas Futuras. Este capítulo plantea la totalidad de las conclusiones de la investigación que se ha llevado a cabo y se apuntan las líneas en las que se puede continuar el trabajo realizado y las nuevas líneas abiertas surgidas en torno a él.

2 Estado del arte

En este capítulo se realiza una revisión de los conceptos y tecnologías relacionadas con la investigación realizada alrededor del problema de la interoperabilidad en entornos de Cloud Computing. El estado del arte se ha dividido en cinco grandes bloques que abarcan la totalidad de la investigación: Cloud Computing, Web Semántica, Servicios Web, Servicios Web Semánticos e Interoperabilidad. El estudio detallado de cada uno de estos bloques proporciona una visión amplia y completa de los trabajos existentes donde se pueden observar sus características, fortalezas y debilidades que servirán como punto de partida de la investigación objeto de esta tesis doctoral. A partir de esto, será posible la construcción del lenguaje orientado hacia la interoperabilidad, la reducción de costes y la automatización, objetivos que pueden suponer un impacto más que significativo en la industria.

2.1 Introducción

El estudio del estado del arte es uno de los aspectos básicos para el desarrollo de cualquier trabajo ya que permite conocer los avances realizados en las áreas de relevancia y establecer los fundamentos sobre los que se asentará la investigación. Partiendo de la información recogida en este capítulo, se podrá establecer con claridad las aportaciones al conocimiento realizadas por esta tesis doctoral.

Por tanto, se puede afirmar que el estado del arte es el resultado de formalizar la información recogida mediante la lectura de la bibliografía específica de los problemas y tecnologías a afrontar, así como de otros temas relacionados con ellos. A partir del análisis de esta información surgirán una serie de ideas y preguntas a las que se intentará dar respuesta, originando el desarrollo de las hipótesis que guiarán el proceso de investigación en su totalidad.

Este trabajo en cuestión gira en torno a cinco grandes áreas diferenciadas: los entornos de Cloud Computing o computación en nube, la Web Semántica y todas sus herramientas y tecnologías asociadas, los Servicios Web, los Servicios Web Semánticos y el concepto de Interoperabilidad. Éste último será el concepto central sobre el que orbitan los demás, ya que supone el problema crítico a resolver mediante el resto de las áreas estudiadas en el capítulo. La relación que se establece entre los diferentes conceptos estudiados se muestra a continuación en la Figura 2:

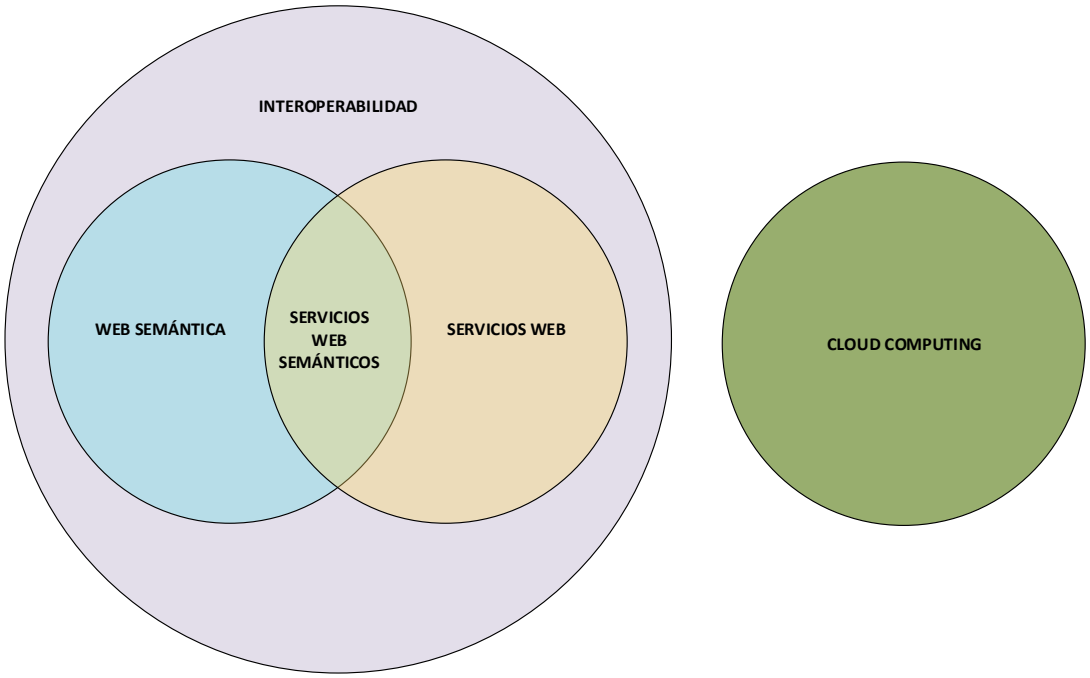


Figura 2. Relación entre los conceptos estudiados en el Estado del Arte

La Figura 2 muestra un diagrama en el que se puede observar como el problema de la interoperabilidad envuelve a los conceptos de Web Semántica, servicios web y servicios web semánticos. Esto es debido a que estas tecnologías nacen con el objetivo de conseguir la interoperabilidad y solucionar los problemas asociados a ella. Además se puede comprobar cómo la intersección de las tecnologías relacionadas con la Web Semántica y los servicios web originan la creación de los servicios web semánticos. Por último, el Cloud Computing aparece como un ente aislado. La razón para ello es que no existe una relación directa entre los entornos de computación en nube y las tecnologías presentadas, de hecho, el Cloud Computing no se puede definir como una tecnología, sino como un paradigma formado por numerosas tecnologías que trabajan conjuntamente para crear un nuevo modelo de computación.

En este trabajo se estudiará la posibilidad de proporcionar un enfoque original en el que el Cloud Computing actúa como entorno en el cual actuarán las tecnologías previamente mencionadas para que, conjuntamente, puedan solucionar los problemas de interoperabilidad que llevan estudiándose desde hace décadas. El estado del arte desarrollado a lo largo de este capítulo presentará los trabajos más relevantes realizados en estas áreas y de esta manera se podrá comprobar si se adaptan a las ideas que se pretenden conseguir y permiten lograr los objetivos fijados para esta investigación.

2.2 Evolución histórica

A menudo resulta difícil situar los diferentes conceptos y tecnologías que forman parte de un estado del arte y comprender la evolución y relación que se establece entre ellos. Esta sección intenta clarificar estos aspectos mediante la presentación de las distintas líneas seguidas por cada uno de los conceptos relevantes para la investigación y explicar el porqué de su surgimiento y evolución.

Este trabajo se centra en la resolución del problema de la interoperabilidad aplicada a los datos almacenados en diferentes aplicaciones heterogéneas. El problema de la interoperabilidad surge hace décadas. El gran crecimiento en el número de sistemas de información y, por tanto, en el número de elementos y componentes relacionados con ellos hace que crezca la heterogeneidad y la necesidad de comunicación entre ellos. La primera referencia de relevancia se encuentra en (Litwin & Abdellatif, 1986), donde los autores tratan de resolver problemas de interoperabilidad en el ámbito de las bases de datos. Desde entonces, la interoperabilidad ha sido una de las metas de los sistemas y modelos de computación y a la vez uno de los grandes problemas en el mundo de la Informática por la dificultad que entraña el conseguir una solución global y aceptable.

Alrededor del año 1991 Tim Berners-Lee comienza a esbozar una de las grandes revoluciones tecnológicas de la Historia: Internet. Esta idea culmina con el conocido artículo (Tim Berners-Lee, Cailliau, Groff, & Pollermann, 1992) y que termina por establecer este sistema de información mundial en 1994. Con el surgimiento de Internet, los usuarios cambian su rol para pasar de ser meros consumidores de información a también creadores de ella. El aumento de la popularidad y, por consiguiente, de usuarios de la Red, han hecho que la información disponible crezca de forma exponencial, complicando las tareas de búsqueda y obtención de la información deseada. Con el objetivo de solucionar este problema nace la Web Semántica (Tim Berners-Lee, 1998), una serie de tecnologías que pretenden aportar conocimiento sobre los datos de manera forman para que sean legibles por las máquinas, que así puedan procesar la información, mejorando la interoperabilidad entre los sistemas y también en la Web. Berners-Lee ya intentó añadir la semántica cuando creo la Web, pero diversos motivos lo impideron (Carvin, 2005). A pesar de ello, no lo olvidó y retomó la idea manteniendo los objetivos que se proponían.

En este momento, existía una fuente de información en continuo crecimiento y un conjunto de tecnologías cuyo objetivo era añadir mejoras (metadatos) a la información existente y contribuir a la consecución de la interoperabilidad. Es entonces cuando irrumpen los servicios web. Su aparición data de algún tiempo antes, pero su eclosión tal y como se conocen hoy en día surge alrededor del año 1999. Se definen como aplicaciones que permiten la comunicación entre sistemas y contribuyen como un paso más para lograr la ansiada interoperabilidad (Brown, Lampson, & Liu, 1998). Estos servicios, permiten añadir una capa de abstracción sobre la Red que los convierte en un conjunto de aplicaciones accesibles desde cualquier punto.

Llegados a este punto parece que se están afrontando los principales problemas de la Web con cierto éxito aunque aún son necesarias algunas mejoras. Diversas líneas de investigación convergen entonces hacia una nueva contribución en pos de la interoperabilidad: los Servicios Web Semánticos que comienzan a tomar forma en (McIlraith, Son, & Zeng, 2001) aunque su gran auge llegara unos años después. Surgen gracias a la unión producida entre los servicios web y las anotaciones provenientes de la Web Semántica y permiten crear servicios denominados "inteligentes". Con ellos se propone definir ontologías de funcionalidad que permitan que los servicios web extraigan cierto conocimiento para automatizar tareas (descubrimiento, composición, etc.) y permitir así aumentar el grado de interoperabilidad. Con su aplicación, estos sistemas inteligentes extraen la información necesaria en cada momento y permiten ahorrar costes y tiempo.

El problema de la interoperabilidad sigue existiendo en nuestros días. En la actualidad un gran número de grupos de investigación se encuentran trabajando en diferentes enfoques que permitan afrontar y solucionar el problema. La Web Semántica sigue orientando sus esfuerzos en esa dirección y nuevas perspectivas como la proporcionada por Linked Data (Bizer, Heath, & Berners-Lee, 2009) siguen avanzando para conseguir el gran objetivo final de una Web completamente interoperable.

El último concepto de relevancia para el desarrollo de esta tesis doctoral sigue una línea paralela a las mostradas anteriormente si bien, posee un claro nexo de unión con los anteriores: Internet. El Cloud Computing surge como la conjunción de varias tecnologías ya existentes como un paradigma que permite acceder a sistemas y servicios a través de Internet. Uno de los primeros artículos presentados al respecto se encuentra en (Tanenbaum & Steen, 2006), en el que se menciona el surgimiento de este paradigma dentro del ámbito de los sistemas distribuidos. La computación en nube fue definida como una revolución debido a que cambiaba por completo el paradigma de software tal y como se había concebido hasta la fecha, pasando a un modelo de pago por uso y software distribuido a través de la Red. Desde su aparición el número de usuarios de estas tecnologías ha crecido enormemente y aún hoy sigue creciendo. Esto es debido a que el alquiler de infraestructuras y la posibilidad de trabajar con plataformas distribuidas desde cualquier lugar suponen un gran ahorro de costes y en tareas de mantenimiento, de las cuales se encarga el proveedor del servicio. Esta realidad hace que sea una opción muy recomendable en todos los ámbitos de la empresa ya que es muy atractivo para nuevas y pequeñas empresas que no pueden hacer frente a grandes inversiones iniciales pero también para grandes compañías cuyos gastos de mantenimiento y seguridad son muy elevados y ven como este modelo puede suavizarlos. Sin duda el Cloud Computing ha irrumpido con fuerza en el mundo de la Informática y parece que aún existe potencial por explotar y aprovechar todas las ventajas que ofrece.

Como se puede observar en la evolución de las tecnologías que forman el estado del arte de esta investigación, el desarrollo de Internet ha influido y resultado aspecto clave en todas ellas. Su surgimiento y el de los conceptos y paradigmas mencionados son fruto de la gran influencia que ha ejercido la World Wide Web en la sociedad de la información. Desde su aparición en los años noventa ha sido sin duda el aspecto que más ha marcado el progreso de la Informática y también de la sociedad en general.

La Figura 3 muestra la cronología de los conceptos clave de este trabajo y presenta gráficamente el momento en el que se encuentran las primeras referencias de cada uno de ellos, aunque el desarrollo global de todos ellos haya sido posterior:

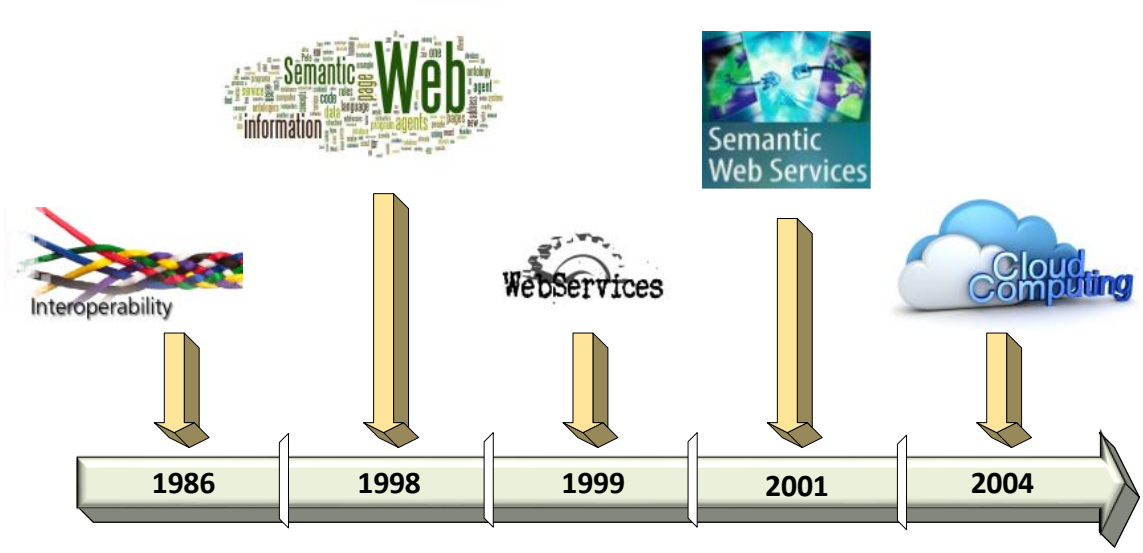


Figura 3. Cronología del Estado del Arte

Como resumen de esta evolución se puede decir que el problema de la interoperabilidad entres sistemas existía desde siempre en los sistemas informáticos, pero sin duda, el surgimiento y posterior auge de Internet acentuaron este problema. Para solucionar esto y los problemas derivados de la gran cantidad de información vertida en la Red, surgió la Web Semántica y, prácticamente de la mano, los servicios web que permitían el intercambio de información entre sistemas. La conjunción de ambas tecnologías permitió la creación de los servicios web semánticos, que otorgaron "inteligencia" a los servicios para automatizar algunas tareas. Diferentes tecnologías y conceptos existentes desde hacía años se agruparon con posterioridad para crear el paradigma del Cloud Computing, cada vez más extendido, y que ofrece servicios a través de Internet de forma barata y sencilla, formando entornos adecuados para ámbitos diversos y cambiando la concepción del software existente hasta ese momento.

La evolución de todos estos conceptos es difícil de predecir. La Web Semántica sigue realizando aportaciones focalizándose cada vez más en las tecnologías de Linked Data, los servicios web continúan empleándose en todo tipo de sistemas y cada día surgen nuevas plataformas e infraestructuras de computación en nube, que ha pasado en pocos años a convertirse en una tecnología madura y ampliamente utilizada. Con todo ello, se puede concluir que las tecnologías empleadas en esta investigación han resultado relevantes para la concepción de la Informática tal y como la conocemos hoy, siguiendo una rápida evolución que ha permitido que en la actualidad sean consideradas como tecnologías punteras en sus respectivos ámbitos.

2.3 Cloud Computing

Para comprender en su totalidad los conceptos relacionados con el Cloud Computing, es necesario clarificar qué es la computación en nube. El concepto de "nube" se utiliza como una metáfora de Internet, como una abstracción de la compleja infraestructura que representa, tal y como aparece en (Armbrust et al., 2009). Por esta razón, como se menciona en (Ograph & Morgens, 2008), podemos afirmar que el Cloud Computing es el paradigma o el modelo de tecnología de la información que ofrece servicios computacionales a través de Internet.

Cloud Computing se presenta a menudo como una tecnología nueva, pero no lo es. Resulta más apropiado presentarlo como un nuevo enfoque que combina tecnologías ya conocidas (R. Buyya, 2009) como por ejemplo sistemas operativos, bases de datos, servidores, redes, multitenancy, middleware, virtualización, herramientas de gestión, etc.

Existe un malentendido común entre la computación en nube y la computación autónoma, computación en grid o Utility Computing. De hecho, el Cloud Computing habitualmente depende de grids para su implementación, tiene cierta autonomía (reacciona cuando la demanda aumenta y reajusta los recursos de manera automática) y se cobra bajo demanda. Sin embargo, el Cloud Computing tiene un mayor alcance como se explica en (Foster, Zhao, Raicu, & Lu, 2008). La esencia del Cloud Computing no se encuentra tanto en las herramientas que usa (software, plataformas, tecnologías...), si no en la forma en la que las usa, agrupa y ensambla, unificando algunos de los principios más importante para la creación de dinámicas apropiadas como se muestra en (Rajkumar Buyya, Yeo, Venugopal, Broberg, & Brandic, 2009).

El siguiente grafico representa la estructura de una nube convencional:

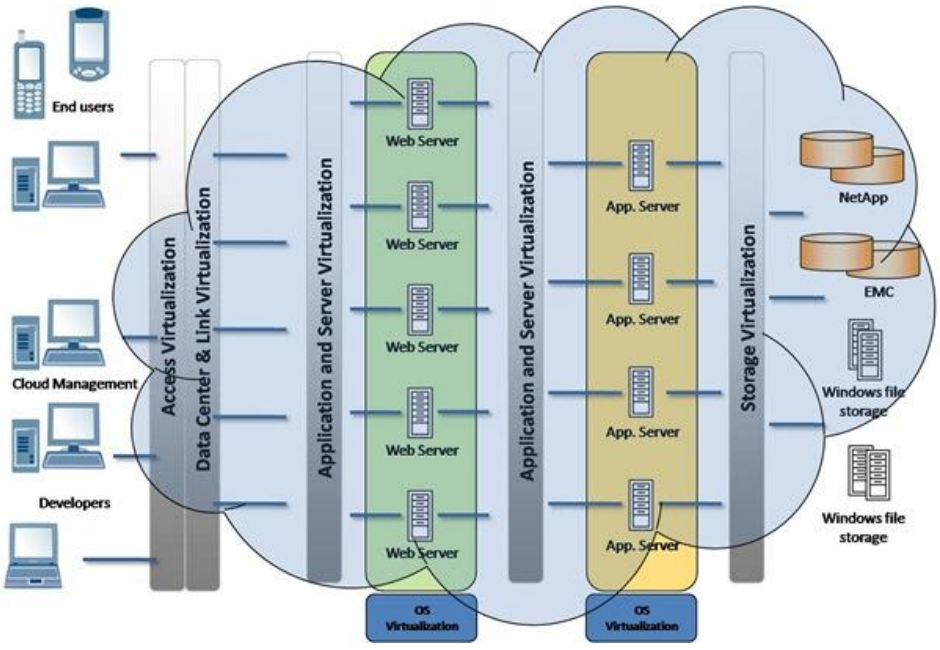


Figura 4. Interior de la nube (DevCentral)

El Cloud Computing se puede dividir en tres niveles (ver Figura 5) en función de los servicios que se encuentran actualmente (Zhang, Cheng, & Boutaba, 2010) y que se comentará en mayor profundidad en el apartado 2.3.2 de este documento. Desde el más interno hasta el más externo se encuentran:

- Infraestructura como servicio (IaaS)
- Plataforma como servicio (PaaS)
- Software como servicio (SaaS)

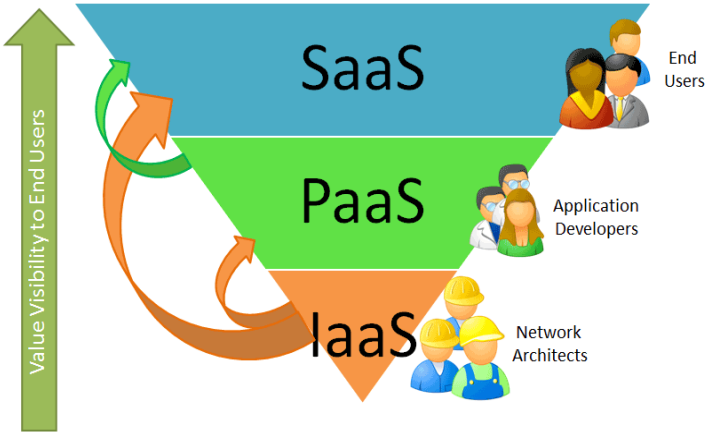


Figura 5. Niveles de Cloud Computing (SaaSMania)

La siguiente imagen completa lo reflejado en la anterior y muestra, a modo de resumen, los principales elementos del Cloud Computing:

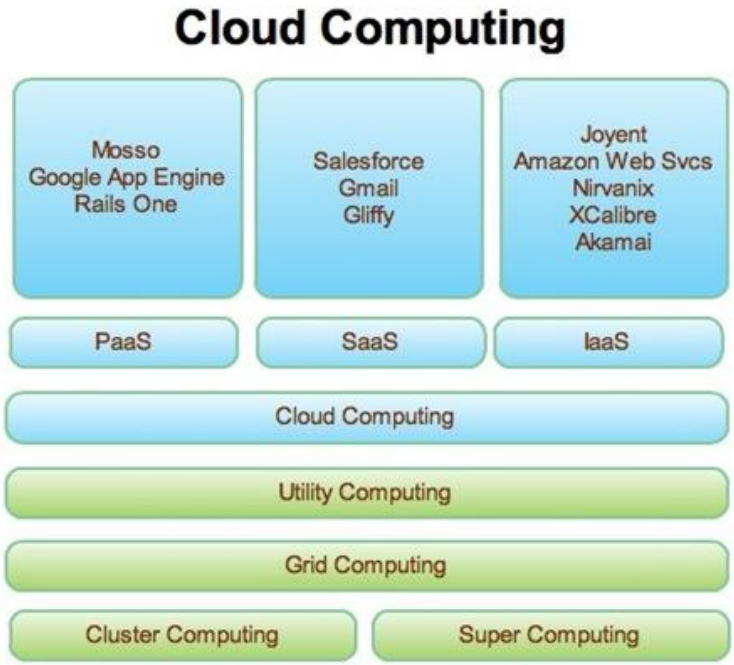


Figura 6. Elementos del Cloud Computing (www.newsandreviews.in)

La Figura 6 muestra en las tres capas superiores los aspectos que se refieren tradicionalmente como Cloud Computing, mientras que en las tres inferiores aparecen algunas de las tecnologías subyacentes que llevaron al desarrollo del paradigma.

2.3.1 Características clave

La principal función de las nubes es emplear los recursos físicos de los que se disponga con el objetivo de proporcionar al usuario una serie de servicios de computación bajo demanda, que sean escalables y que, muy frecuentemente, se presentan en un entorno multiusuario (multitenant) (Vouk, 2008). Esta definición abarca las características clave de Cloud Computing las cuales se describen a continuación de una forma más detallada:

- **Las TI como servicio**. En primer lugar, todo lo que Cloud Computing es capaz de ofrecer, lo ofrece como servicio. Eso significa que los usuarios pueden consumir esos servicios a través de la nube sin necesidad de conocer el manejo de las complejas estructuras subyacentes o los recursos físicos que usan. Esta abstracción ofrece a las empresas la oportunidad de disminuir costes en equipos hardware y personal técnico encargado de su mantenimiento.
- **Escalabilidad bajo demanda**. En segundo lugar, esos servicios se ofrecen bajo demanda, siguiendo el modelo de pago por uso. Los recursos se asignan dinámicamente para adaptarse a las necesidades de los usuarios en cada momento. Desde el punto de vista del consumidor, dichos recursos parecen ilimitados y pueden ser adquiridos en cualquier cantidad en todo momento.
- **Independencia del dispositivo y localización**. Además, los servicios son accesibles a través de Internet, por lo que los usuarios pueden acceder desde distintas localizaciones y dispositivos (PCs, PDAs, etc.).
- **Pago por uso**. La computación se ofrece como un servicio medible y facturable, de forma similar a la electricidad, el agua, o la telefonía.
- **Entorno multi-usuario**. Por último, permite compartir recursos y costes entre gran número de usuarios. Este hecho deriva en la mejora del proceso de negocio ya que las empresas comparten datos y aplicaciones, centrándose en el proceso de negocio en lugar de las infraestructuras y tecnologías que lo soportan. Al mismo tiempo, los usuarios tienen en todo momento a su disposición las últimas versiones de los sistemas y servicios que usan. El modelo Multitenant (una única instancia compartida por múltiples usuarios) hace que a los proveedores les resulte mucho más fácil ofrecer acceso instantáneo a las nuevas características a todos los usuarios, en comparación con los modelos tradicionales.

2.3.2 Modelos de prestación de servicio

Como se ha comentado en la sección 2.3, se pueden distinguir tres modelos de prestación de servicio, que hacen referencia a modelos de negocio de software, entornos y hardware donde las empresas ofrecen el servicio pagado del uso de sus capacidades (Data Centers, Web Services, etc):

- **Software como Servicio (SaaS)**. Es el más conocido de los tres modelos y el que suele tener como objetivo al cliente final, aquel que utiliza el software para ayudar, mejorar o cubrir algunos de los procesos de su empresa. El SaaS es aquella aplicación "consumida" a través de Internet, casi siempre a través del navegador, y donde tanto la lógica de la aplicación como los datos residen en la plataforma del proveedor.
- **Plataforma como Servicio (PaaS)**. Se puede definir como un conjunto de plataformas compuestas por uno o varios servidores de aplicaciones y una bases de datos (no todas la plataformas incluyen la posibilidad de tener base de datos) que ofrecen la posibilidad de ejecución de aplicaciones (escritas en java, ruby, python, etc.). El proveedor será el encargado de escalar los recursos en caso de que la aplicación lo requiera, del rendimiento óptimo de la plataforma, seguridad de acceso, etc.
- **Infraestructura como Servicio (IaaS)**. Se incluyen en este nivel todos los servicios de computación que utilizan virtualización para ofrecer máquinas virtuales con diseño específico. Dichos servicios vienen acompañados por los de almacenamiento relacionado (bases de datos) y no relacionado (discos).

La Figura 7 muestra de manera gráfica los tres modelos descritos:

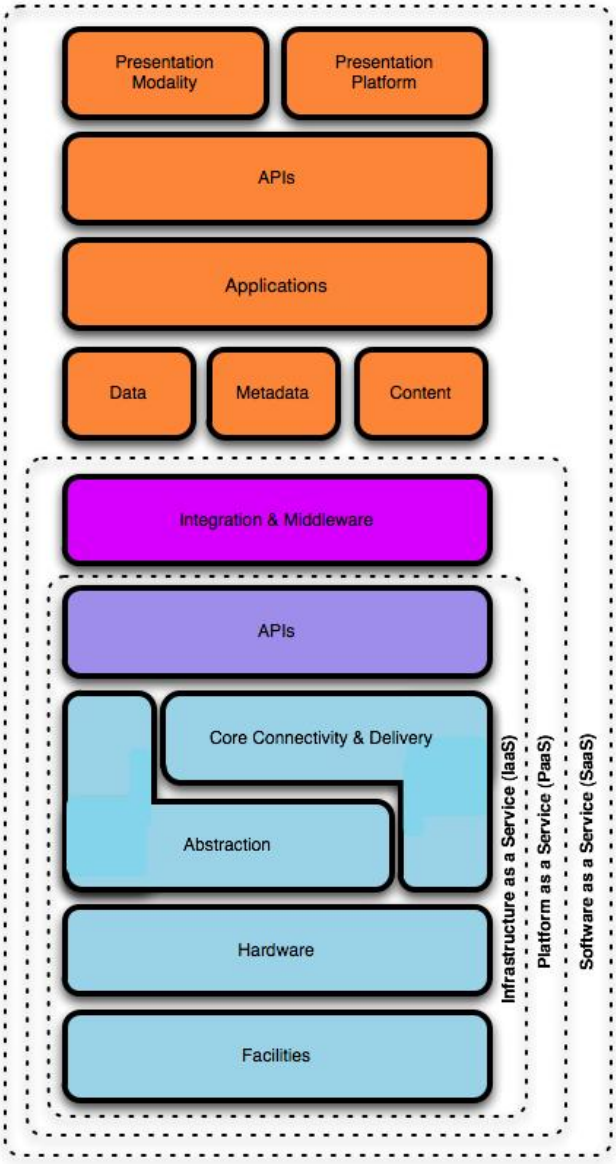


Figura 7. Relación entre modelos de prestación de servicio (www.rationalsurvivability.com)

Ninguno de estos modelos existe en un entorno aislado, por lo que es importante entender la estrecha relación entre ellos. IaaS es la base de todos los servicios en la nube, con PaaS articulado encima de IaaS y SaaS, a su vez, articulado sobre IaaS.

La Figura 8 ilustra la taxonomía de las soluciones disponibles (Rimal, Choi, & Lumb, 2009) dentro del Cloud Computing, según los distintos modelos explicados:

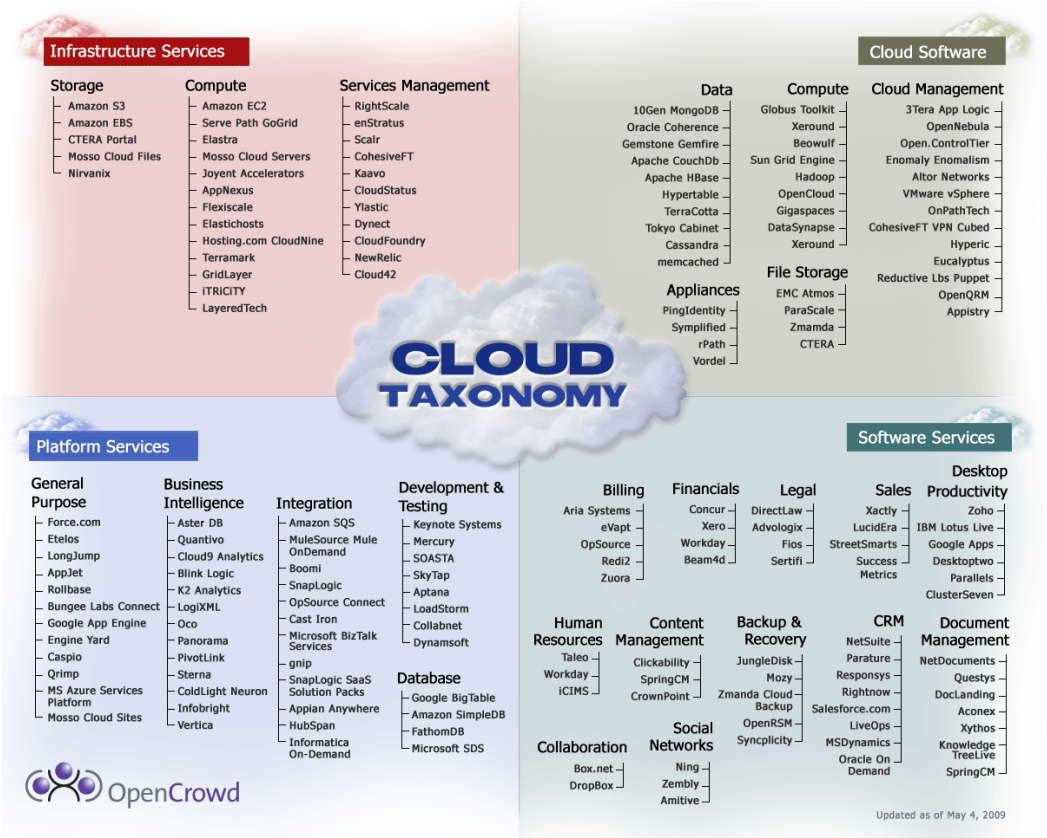


Figura 8. Taxonomía de servicios de Cloud Computing (www.opencrowd.com)

Para la investigación presentada resultan de especial interés los dos niveles superiores de prestación de servicio (SaaS y PaaS). Por este motivo se describen con mayor nivel de detalle en la siguiente sección.

2.3.2.1 Software como Servicio (SaaS)

Software-as-a-Service (SaaS) (Campbell-Kelly, 2009) es un término que se refiere a un modelo de distribución donde el suministrador del servicio proporciona, además del propio software, otros servicios como mantenimiento, ayuda y soporte. La principal ventaja es que el software requerido se distribuye y aloja en la red. Gracias a esto, el usuario no necesita instalar este software. Otros aspectos como la seguridad, calidad del servicio y el rendimiento son completamente gestionados por el suministrador. En otras palabras, el cliente tiene su sistema alojado en la compañía de tecnología de la información (Knorr & Gruman, 2008). Debido a esto, en áreas funcionales como la gestión de los recursos humanos, el paradigma SaaS continúa creciendo por encima de las previsiones (Galinec, 2010).

Esto supone un cambio radical en la forma en la que se desarrollan los modelos de distribución abiertos para la industria de las TIC europeas, cuyo modelo apoya a la industria del software, transformando el modelo de negocio de desarrollo dentro de la empresa a desarrollo de

aplicaciones de Software-como-Servicio. Este cambio en la concepción de la distribución de software conlleva importantes ventajas como la alta escalabilidad, el drástico descenso de los costes y una gran eficiencia en términos de rendimiento y disponibilidad para los sistemas de información corporativos de cualquier segmento del mercado (Armbrust et al., 2009). Otro aspecto interesante es que la utilización de la aplicación se lleva a cabo en servidores centralizados en lugar de estar alojados en localizaciones del cliente, el cual accede a todos los servicios a través de la Web. Se trata por tanto de un modelo que une el producto y el servicio para proporcionar al usuario, sea cual sea, una solución completa que le permita optimizar sus costes y sus recursos.

Salesforce, compañía puntera en este campo, lo define así: *"El software como servicio (o SaaS) es una forma de ofrecer aplicaciones a través de Internet como si de cualquier otro servicio se tratara. En vez de instalar y mantener el software, sólo tiene que acceder a él a través de Internet, con lo que evita tener que gestionar el complejo software y hardware"*. ¿Qué implicaciones tiene esto en primera instancia? El cliente no necesita licencia alguna para utilizar el software, no necesita ningún soporte físico ni servidores y tampoco mantener el software o instalarlo. El único elemento necesario para utilizar este tipo de sistemas es un navegador web para acceder al software y ejecutarlo a través de Internet. La empresa que ofrece el servicio se encargará de todos los aspectos mencionados anteriormente.

Para una mejor comprensión de las implicaciones de este paradigma se propone el siguiente ejemplo. Hasta ahora si se deseaba tener una aplicación el proceso consistía en la adquisición de un soporte físico (por ejemplo CD) o la descarga o consecución de algún tipo de ejecutable, todos ellos con sus respectivas licencias para ejecutar en la memoria del propio ordenador. Entonces una empresa con sede en Madrid compra un software y debido a su buen funcionamiento decide que ese software se utilice también en su delegación de Sídney (Australia). Para ello necesitaría conseguir nuevas licencias, apropiarse de ese software mediante el envío por correo o envío electrónico, etc., esto siempre en el caso de que el software fuese configurable y fácilmente instalable. Si no fuese el caso, ¿cuál sería la solución? ¿Enviar el equipo de diseñadores y programadores que realizaron el proyecto en Madrid a la delegación de Sídney? Además, ¿qué pasaría si la aplicación es muy potente y necesita máquinas potentes para el proceso de datos, servidores...? Los costes de desarrollar este software se duplicarían. El paradigma SaaS permite ahorrar estos costes además de facilitar cada una de las tareas que se deben realizar en cada caso.

El caso más famoso de compañía que trabaja orientada hacia este paradigma es Salesforce (Salesforce, 2011) que ofrece aplicaciones a través de su plataforma de Cloud Computing y que se encuentra al frente de este movimiento desde sus comienzos, proporcionando diferentes servicios en función de las necesidades de los clientes. A raíz del auge de esta tecnología surgieron muchas más empresas dedicadas a SaaS como es el caso de Netsuite (Netsuite, 2012), Concur (Concur, 2012), Jamcracker (Jamcracker, 2012), etc. Un interesante estudio económico de SaaS se presenta en (L.M. Vaquero, Rodero-Merino, Caceres, & Lindner, 2008) donde todas las implicaciones de las inversiones se encuentran claramente expuestas.

2.3.2.1.1 Características Generales

Para entender de una manera más amplia el nuevo modelo de desarrollo, se presentan a continuación las características detalladas del mismo:

- La primera consecuencia del modelo SaaS es la desaparición del concepto de licencia. A partir de este momento se introduce la idea de pago por uso. De esta manera los clientes realizan suscripciones a determinados servicios para poder utilizar las aplicaciones que desean.
- El software se distribuye a través de la red.
- Se convierte en un modelo descentralizado de uso de aplicaciones software.
- La escalabilidad aumenta hasta llegar a ser "ilimitada".
- La importancia del modelo reside en el servicio, dejando en segundo plano la tecnología, aplicaciones y demás aspectos técnicos.
- La aplicación se encuentra alojada. Esto implica que se proporciona el servicio a un gran número de clientes y que se trata de una infraestructura pública que hace que varias empresas o clientes puedan suscribirse al servicio proporcionado.
- Los clientes pasan a ser entidades virtuales.
- Se construyen plataformas de N clientes.
- El cliente obtiene los siguientes beneficios:
    - Disminución del riesgo.
    - Reducción del coste y de la inversión inicial.
    - Alta escalabilidad
    - Se focaliza en el negocio.
    - Aumento de la seguridad, a pesar de la desconfianza de no conocer de forma exacta la gestión de sus datos.
    - Rápida respuesta y adaptación a los cambios.

2.3.2.1.2 Características Tecnológicas

Un aspecto muy a tener en cuenta en este campo son las implicaciones y consideraciones tecnológicas que hay que cubrir cuando se pretende ofrecer soluciones basadas en SaaS.

Existen varias perspectivas desde las que estudiar estas consideraciones. A continuación se presentan estos diferentes puntos de vista y se detallan los aspectos más significativos a tener en cuenta en cada uno de ellos.

2.3.2.1.2.1 Fabricante de software

La perspectiva del fabricante de software debe variar y modificar la visión clásica de construcción de aplicaciones que se tenía hasta el momento. Las principales consideraciones que se introducen son las siguientes:

- En primer lugar, se hace necesario volver a escribir parte o la totalidad del código de las aplicaciones, de forma que se permita su ejecución en entornos descentralizados en los que participen números elevados de usuarios, de empresas…
- Se deben incluir servicios que no formaban parte de las aplicaciones tradicionales. Ejemplos de estos servicios pueden ser la monitorización o la facturación.
- Inclusión de un plan de mejoras continuas y nuevas versiones que se adapten a las nuevas situaciones y solucionen problemas existentes.
- Gestión de la escalabilidad y los recursos en la medida en que la infraestructura de alojamiento puede ser compartida por distintas aplicaciones y un número indeterminado de usuarios.
- Ofrecimiento de soporte 24x7. Es necesario asegurar que las aplicaciones funcionan de forma adecuada en todo momento, eliminando las anomalías que surjan en el servicio ofrecido.

  Es básico el aseguramiento de la seguridad en todas y cada una de las transacciones y en el almacenamiento de la información.

2.3.2.1.2.2 Arquitectura de la solución

Para establecer las bases sobre la que se asentará la aplicación es condición necesaria que se cumplan una serie de requisitos:

- **Escalabilidad de manera natural**. Para reaccionar de forma adecuada ante el creciente aumento de clientes es necesario definir un plan de escalabilidad de la aplicación
  - Mecanismos de caching
  - Compartición de recursos
  - Entrada/Salida asíncrona

  Además se deben asegurar unos tiempos de respuesta adecuados, teniendo en cuenta que en SaaS no se posee una previsión de los clientes/usuarios que tendrá la aplicación.
- **Configurable**. El usuario debe ser capaz de elegir y personalizar lo que será su modelo de aplicación.
- **Eficiencia**. La solución ha de funcionar de acuerdo a lo establecido y haciendo un uso adecuado de los recursos.

Además de los requisitos mencionados, a la hora de construir la arquitectura de la aplicación, se deben considerar los siguientes factores:

- **Seguridad**. El modelo SaaS está pensado para almacenar gran cantidad de datos de diferentes clientes. El aseguramiento de esos datos así como la eficiencia en gestionarlos son aspectos fundamentales.
- **Arquitecturas SOA (Service Oriented Architecture).** Se encargan de conseguir soluciones no aisladas, o lo que es lo mismo, que puedan interoperar con otras aplicaciones o servicios.
- **Personalización y configuración de la aplicación a medida de cada cliente**. Es importante subrayar que esto se hace utilizando la misma solución. Se trataría de compartir infraestructura y utilizar metadatos para habilitar la personalización y configuración que exige una solución multicliente y multimarca.
- **Escalabilidad a diferentes niveles**:
    - De aplicación
    - De estructura de datos (esquemas de replicación, mantenimientos de referencias al cambiar estructuras de datos, etc.)
    - Pruebas de stress. Es fundamental hacer este tipo de pruebas teniendo en cuenta el modelo de aplicaciones alojadas en el que se basa.
    - Capacity planning. Previsión de un plan de incremento de capacidad para prever situaciones de incremento de carga.
- **Configurable a diferentes niveles:**
    - Esquema de datos
    - Procesos (workflow) y reglas de negocio
    - Interfaz de usuario

2.3.2.1.2.3 Avanzar hacia SaaS

Los apartados anteriores han proporcionado una idea global de las diversas consideraciones que se deben manejar a la hora de construir un sistema SaaS. Aún así, existen ciertas preguntas inevitables que conviene aclarar y responder antes de comenzar la andadura hacia SaaS.

- **¿Cómo se avanza hacia SaaS?**

  Es imprescindible que para la realización de un sistema basado en el modelo SaaS se desarrolle un diseño de la solución que recoja los siguientes aspectos:

    - Ha de ser escalable
    - Debe permitir un número ilimitado de usuarios
    - El diseño no debe afectar a otras soluciones
    - Fiabilidad
    - Integración de los servicios y funcionalidades necesarios en una solución SaaS

- **¿Qué tipo arquitectura es más conveniente?**

  Para el proceso de elección y desarrollo de la arquitectura se debe contemplar lo siguiente:

  - Tecnologías a utilizar. Java, .NET, tecnología para la interfaz gráfica de usuario (GUI), etc.
  - Estudiar el número de capas adecuado
  - Observar el Sistema Operativo de los servidores
  - Permitir integración de otras aplicaciones del cliente

- **¿Sólo son posibles aplicaciones web dentro de SaaS?**

  Definitivamente no. Se pueden contemplar escenarios offline y clientes que utilicen las funcionalidades de los servicios alojados.

- **¿Qué ocurre con los dispositivos móviles?**

  Este tipo de sistemas son ideales para dispositivos móviles y se contemplan como parte de la solución.

- **Una solución Saas, ¿está aislada o se puede integrar con otras aplicaciones y soluciones?**

  No debe ser una solución aislada. Se tienen que incluir los puntos de extensibilidad necesarios para que se pueda integrar con otros entornos.

Además de las respuestas a estas preguntas, es necesario afirmar que una formación técnica adecuada es muy importante a la hora de desarrollar los modelos. También presenta especial relevancia la creación de planes de contingencia para el outsourcing con el fin de evitar problemas que puedan surgir posteriormente.

Con el fin de completar la información, se presenta la siguiente tabla que refleja las principales diferencias entre las aplicaciones tradicionales y las aplicaciones tipo SaaS:

	Modelo de Aplicaciones SaaS	Aplicaciones Tradicionales
Acceso al sistema desde cualquier parte del mundo	Sólo precisa de Internet para tener **acceso 100% on-line** a la información de la empresa	Sólo puedo acceder a la información desde la propia empresa
Escalabilidad del sistema (Pago por uso)	El cliente paga por aquellos módulos que necesita. Se adapta al crecimiento de la empresa sin tener que cambiar de aplicación	No son escalables. El cliente paga por toda la aplicación. No soporta el crecimiento de la empresa

Coste de mantenimiento	No es necesario contratar mantenimiento, está incluido en la mensualidad.	Es necesario contratar adicionalmente un mantenimiento.
Coste inicial de la inversión	Mínimo	Requiere una inversión inicial elevada (pago de licencias, hardware…)
Modo de implantación	Implantación muy poco agresiva.	La implantación requiere pasos radicales.
Necesidades del hardware	Requiere unas prestaciones mínimas de hardware conseguir un rendimiento satisfactorio.	Requiere una inversión elevada en hardware para obtener un buen rendimiento.
Obsolescencia del hardware	El hardware nunca se queda obsoleto por las necesidades de la aplicación.	El hardware se queda obsoleto por lo que requiere nuevas inversiones.
Integración de los procesos de la empresa	Integración absoluta. La **plataforma** engloba todas las áreas de la empresa.	Integración complicada y con elevado coste.
Adquirir actualizaciones	No es necesario comprar actualizaciones. La aplicación está siempre **100% actualizada**.	Es necesario y supondrán un coste adicional.
Copias de seguridad	Garantizadas. Se realizan diariamente, en distintos servidores e incluso ubicaciones, sin preocupaciones por parte de la empresa.	Es necesario adquirir un sistema de copias de seguridad. Deben ser gestionadas por la propia empresa.
Personal especializado para administrar el sistema	Con la cuota mensual dispone de un servicio de profesionales dedicado a su sistema.	Necesita invertir en personal interno o externo para administrar su sistema.
Tiempo de puesta en marcha	Mínimos.	Pueden llevar a varios meses dependiendo de la empresa.
Tiempo de retorno de la inversión	Tiempo muy reducido.	Tiempo elevado, motivado entre otras cosas por el coste inicial del software, coste en hardware necesario y tiempo de puesta en marcha.

Conectar delegaciones geográficamente dispersas	Sólo necesito internet para tener acceso 100% on-line a la información de mi empresa desde cualquier ubicación.	Requiere una inversión en infraestructura y mantenimiento.
Crear tienda virtual integrada en el sistema	Muy sencillo. Todo integrado y en un mismo sistema.	Requiere aplicaciones externas e interconexión entre las mismas.
Controlar stocks y almacenes en tiempo real	Es posible, en tiempo real e independientemente de su ubicación.	No es posible. En caso de serlo requieren una elevada inversión tecnológica.
Intercambiar documentos electrónicamente	Es posible, emitiendo un código unívoco de identificación del documento en la plataforma.	No es posible.
Integración con servicios de asesoría: laboral, contable y fiscal	100% integrado. Consiguiendo comodidad, rapidez y eficacia.	No existe integración.
Espacio para datos	No existe límite.	Espacio limitado al sistema de almacenamiento de la empresa

Tabla 1. Comparativa Saas vs Software bajo licencia

2.3.2.1.2.4 Ventajas

Mediante el uso de la tecnología SaaS y con un punto de conexión a Internet se puede agregar movilidad a los sistemas. Esto representa una gran ventaja ya que cada vez son más las actividades de la empresa que se realizan fuera del ámbito de la misma y por lo tanto, lejos de donde están ubicados los sistemas de información de su compañía (Ju, Wang, Fu, Wu, & Lin, 2010). Hoy en día las empresas necesitan disponer de la información relativa a sus negocios en cualquier momento, desde el lugar que quieran y en la forma que prefieran.

Por ello el acceso desde cualquier lugar, en cualquier momento y la actualización en tiempo real son factores claves que aporta el software SaaS. Estas características permiten las siguientes ventajas:

- **Uso más eficiente de los recursos de la empresa**: se evitan redundancias de información.
- **Uso más eficiente del tiempo**: la información está actualizada en tiempo real, con lo que se consiguen decisiones más exactas y en menos tiempo.
- **Reducción de riesgos**: derivados de la mejora en la toma de decisiones.

- **Eliminación de comunicación y explicaciones innecesarias**: gracias al acceso directo a la información en tiempo real, no es necesario llamar para saber. La comunicación directa entre el hombre y la máquina evita intermediarios, errores "humanos" en la comunicación, problemas de privacidad…
- **Posibilidad de incorporar tecnologías avanzadas**: por ejemplo, los servicios de localización permiten aplicar procesos de optimización de rutas, modificación dinámica en respuesta a eventos, etc.
- **Más y mejor información y atención a los clientes**: se presentan siempre los datos actualizados, sin posibilidad de errores.
- **Visión única de la información de la empresa**: las reglas de negocio son únicas y están centralizadas en el servidor de la aplicación. Un cambio en el sistema tiene efecto inmediato en todos los recursos que acceden a él. No hay procesos de consolidación, lo que supone una mejora crítica en la calidad de la información.
- **Facilita cambios en el negocio**: se cambia la noción de procesos periódicos por la de "services on demand". Actividades tales como el rebaje de stocks o la previsión de pagos venían desarrollándose en procesos periódicos, lo que conllevaba el riesgo de información poco fiable durante los períodos intermedios.
- **Conectividad:** Es necesario asegurar siempre la conectividad para recibir y enviar los cambios de información que se produzcan instantáneamente.
- **Interactividad:** Es necesario el envío de información por parte de todos los componentes de la solución, y que este se produzca rápidamente sin grandes tiempos de retardo.
- **Copias de seguridad:** Son muy simples. Al contratar el servicio de alojamiento con otra empresa, este servicio también será externalizado por lo que no supone una preocupación propia.

En general, y para resumir, se simplificarán todos los procesos y se abandonará la dependencia de un hardware concreto en una localización concreta.

Ventaja de costes

En principio una de las grandes ventajas de SaaS es su bajo coste. A continuación se explicará bajo qué cimientos se sostiene la posibilidad de mantener esos costes por debajo de los alcanzados en los modelos anteriores.

La utilización de SaaS genera economías de escala derivadas del aprovechamiento eficiente de los recursos hardware y humanos del proveedor del servicio y termina repercutiendo en el precio que paga el cliente (Lu & Sun, 2009). Algunos lo han denominado el "low cost" del hardware y el software porque permite acceder a través de Internet a los recursos demandados de una forma óptima con unos costes finales más bajos que la oferta de software y hardware tradicional.

Ha lugar notar que el software bajo licencia tiene otros costes añadidos además del pago de la licencia, estos costes son los siguientes:

- **Costes hardware**: es un hecho común que se necesite comprar máquinas o añadir el software a los servidores que se posean con el fin de gestionarlos. Esta es una misión crítica que necesitará tiempo y el tiempo es dinero.
- **Costes adicionales de software**: probablemente se necesite software como sistemas operativos, bases de datos…
- **Costes de implementación**: en muchas ocasiones se necesita utilizar recursos y tiempo para instalar el software, integrarlo, programar los servers, configuraciones…

Para observar las diferencias de un caso real (ReadWriteWeb), se presenta a continuación un estudio de los costes de implementar una solución en software bajo licencia frente a una solución SaaS:

Software bajo licencia	SaaS ($850 x mes x 100 usuarios)
• Año 1 Servidor MOSS = $4,500 Licencia de acceso de usuario = $90 Almacenamiento y mantenimiento = $5,000 Implementación y soporte de desarrollo = $20,000 **Total = $29,590** • Año 2 y siguientes Almacenamiento y mantenimiento = $5,000 Soporte de desarrollo = $3,000 **Total = $8,000**	• Año 1 Tasa de SaaS = $10,200 Soporte de implementación = $10,000 **Total = $20,200** • Año 2 y siguientes Tasa de SaaS = $10,200 **Total = $10,200**

*La aplicación SaaS tiene el 80% de las funcionalidades de Sharepoint

Tabla 2. Comparativa costes software con licencia vs SaaS (Majal)

Según esto el software bajo licencia tardaría 4 años y medio en ser más barato que el software SaaS aunque esto es un mero espejismo debido a que antes de 4 años y medio con casi total seguridad haya habido que adquirir un software nuevo o renovar el anterior, por lo que habría que comprar nuevas licencias, mientras que una de las ventajas del software SaaS es que se va actualizando.

La situación más genéricamente se observa en el siguiente gráfico desarrollado por Salesforce.

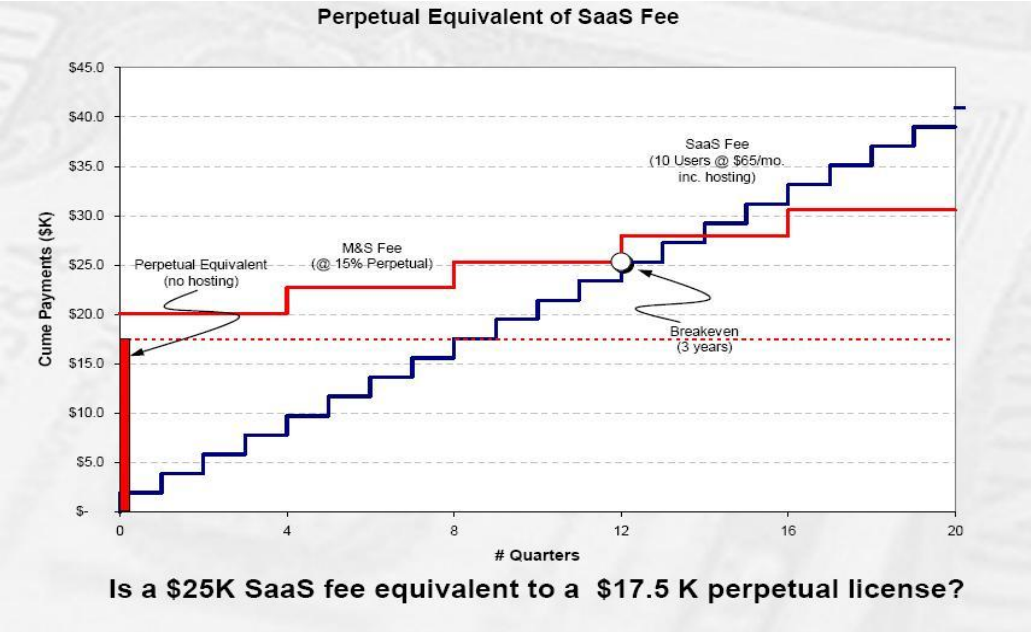


Figura 9. Evolución costes SaaS vs sistema bajo licencia (Salesforce)

En la Figura 9 el M&S Fee representa la cuota por mantenimiento y soporte y el salto entre el coste de la licencia perpetua y el comienzo de la curva roja es el 15% de M&S correspondiente al primer año.

Este gráfico tipo se puede encontrar en muchos análisis de costes de SaaS. De él se pueden sacar conclusiones similares a las dadas en el ejemplo de la Tabla 2. En el caso que nos ocupa, se puede ver como el punto de equilibrio entre SaaS y el software de licencia se obtiene en el tercer año de adopción de la soluciones. Aún así existe una gran diferencia entre ambas curvas y es que en la curva roja no se está teniendo en cuenta el coste de infraestructura (CPD, hardware y personal de mantenimiento) y la renovación de esta infraestructura a lo largo de los años. Si se tuvieran en cuenta estos costes, el punto de equilibrio se desplazaría al menos hasta el 6º o 7º año, pero mucho antes de ese momento es posible que se empezara a pensar en la renovación del hardware, con lo que no se llegaría a obtener un ahorro.

Por tanto si la decisión de compra pasa por la solución más económica y con menor riesgo asociado y a esto se une el resto de ventajas que se obtienen consumiendo Software como Servicio en vez software “in house”, hace que SaaS sea la mejor opción para el desarrollo de determinadas aplicaciones.

2.3.2.1.2.5 Inconvenientes

A pesar de las numerosas ventajas que ofrece el paradigma SaaS, existen también una serie de aspectos que pueden provocar cierta incertidumbre o susceptibilidad (Janssen & Joha, 2011). Los más significativos se presentan a continuación:

- El primer asunto que puede sembrar dudas es la dependencia de Internet. En gran parte este problema viene dado por la falta de infraestructuras en algunos casos, si bien es un problema salvable. Además a medio plazo y gracias a la introducción del WiMAX (que comienza a utilizarse) y en un futuro más lejano con la posibilidad de tener Internet vía satélite, quedaría completamente resuelto.
- La centralización puede ser otro de los inconvenientes, debido a la posible aparición de problemas y averías. Para evitar estas situaciones la mayoría de los servidores incorporan elementos que los hacen tolerantes a fallos (más de una fuente de alimentación, más de un disco duro y comunicación entre ellos por diferentes vías, etc.). Si se contrata el servidor por medio de una empresa de alojamiento, ésta será la que proporcionará esos medios, de forma que los riesgos serán los mismos o menores (al estar controlados por especialistas del campo) que utilizando un servidor tradicional. Además, en el caso que nos ocupa, es precisamente esta característica la que produce la oportunidad para conseguir la interoperabilidad.
- El nivel de confianza en la seguridad de los datos desciende. Los datos de la empresa, sean críticos o no, son privados y el hecho de no estar localizados en sus propias paredes es algo que no suele agradar a las direcciones conservadoras y escépticas. El responsable de TI debe trabajar para cambiar esa mentalidad introduciendo progresivamente aplicaciones de software como servicio que manejen datos no críticos e ir cuantificando las ventajas del SaaS por medio de un cuadro de mando. De todas formas, es importante que el cliente aborde ciertas cuestiones: ¿Cómo de seguros están los datos? ¿Seré capaz de bajarlos? ¿Estarán disponible de forma adecuada y de forma segura? ¿Pueden ser vendidos? ¿Puede cualquier otro alojar la aplicación y mis datos? ¿Los ficheros fuentes de la aplicación son abiertos y por tanto pueden ser alojados con otro proveedor? Las historias de compañías que desaparecen no son infrecuentes, y no únicamente compañías SaaS, también compañías de software tradicionales. La única diferencia es que cuando una compañía de software tradicional cae, la mayor pérdida son los años de soporte que esperabas del vendedor. Cuando un proveedor SaaS cae, las implicaciones son más profundas al tener que considerar los datos vitales del negocio.

  La integración con el resto de aplicaciones del sistema. La situación más común es la de tener aplicaciones con instalación local y SaaS. Por ello, existe un aumento de la complejidad cuando se desea conectar o explotar la información existente en la nube con los datos almacenados en la empresa.
- Sensación de cautividad del cliente. En general, existen los mismos problemas en instalaciones internas y en SaaS, donde los volúmenes de información son importantes y si se añade la latencia y la velocidad de Internet, puede ser determinante para la elección final del software.
- Posibles incumplimientos de los acuerdos sobre el nivel del servicio. Este aspecto está directamente relacionado con el grado de confianza depositado en el proveedor de software, ya que se pone en sus manos el funcionamiento y servicio de la aplicación.
- Al tratarse de un servicio de retención a cuenta, los cargos del usuario se acumulan rápidamente. Limitar el acceso es un método realista de administración efectivo ya que probablemente no todo el mundo necesite acceso, incluso en oficinas de menor tamaño.

- Enfoque a la satisfacción del cliente. Se considera un inconveniente desde el punto de vista del proveedor. Los proveedores SaaS necesitan enfocarse a la satisfacción del cliente o los perderán. Existe la necesidad de ganarse el negocio y confianza de sus clientes mes a mes o ellos simplemente lo dejarán. En contra de los desarrollos tradicionales, los cuales son muy costosos y consumen mucho tiempo, si sus clientes no están satisfechos con el servicio pueden dejarlo en cualquier momento con un coste mínimo.
- Duros procesos de desarrollo. Existen determinados aspectos, como pueden ser la separación de los usuarios, el aprovisionamiento y la escalabilidad que podrían ser complejos de abordar y que obviaríamos en aplicaciones internas. La preparación y el talento de los desarrolladores debe ser mayor y también es mayor la dificultad de encontrar esas cualidades. Además, no existen herramientas de desarrollo específicas que faciliten esta labor.
- Siguiendo con inconvenientes desde la perspectiva de los proveedores, se tiene la problemática de mantener operaciones cuando los ingresos son pequeños. En sistemas tradicionales se manejan cantidades por adelantado que proporcionan un margen de financiación, mientras que ahora la situación varía y los ingresos son más pequeños inicialmente. El proveedor necesitará tener una buena estrategia sobre cómo compensar a su fuerza de ventas al mismo tiempo que pueda disponer de financiación para hacer funcionar la compañía.
- El éxito alcanzado puede resultar problemático. Si el proveedor no se encuentra lo suficientemente preparado puede ocurrir que la situación salga fuera de su control si no tiene la arquitectura apropiada para resolver cuestiones como la escalabilidad.

2.3.2.2 Plataforma como Servicio (PaaS)

El PaaS es uno de los niveles de prestación de servicio del paradigma del Cloud Computing que proporciona una plataforma de computación y un conjunto de componentes y subsistemas como servicio (Chang, Abu-Amara, & Sanford, 2010). También puede ser definido como el conjunto de plataformas compuestas por uno o más servidores de aplicaciones y bases de datos (aunque no es condición necesaria la existencia de una base de datos en la plataforma) que ofrece la posibilidad de ejecutar aplicaciones. Con todo ello se proporcionar una manera de alquilar hardware, sistemas operativos, almacenamiento y redes a través de Internet, permitiendo la adquisición de servidores virtualizados y sus servicios asociados para ejecutar aplicaciones existentes.

En este modelo el consumidor controla el despliegue del software y los parámetros de configuración. El proveedor será el responsable de escalar los recursos cuando sea necesario, proporcionar almacenamiento, mantener un rendimiento óptimo de la plataforma, asegurar la seguridad de la misma, etc.

A diferencia de los otros modelos, el consumidor crea el software empleando las herramientas y librerías proporcionadas por el proveedor del servicio. De esta manera se ofrece un sencillo despliegue de aplicaciones que hacen disminuir los costes y la complejidad de adquirir y

gestionar todo la capa de hardware y software interno y sin la necesidad de proveer de las capacidades de red. Entre las capacidades que ofrece PaaS se encuentran:

- Diseño de aplicaciones
- Desarrollo de aplicaciones
- Testing
- Despliegue
- Servicios:
    - Entornos colaborativos
    - Integración de servicios web
    - Integración de BBDD
    - Securidad
    - Escalabilidad
    - Almacenamiento
    - Persistencia
    - Versionado de aplicaciones
    - Gestión de estados
    - Etc.

Además, este modelo proporcionar ventajas para los desarrolladores. Con PaaS las características de los sistemas operativos pueden ser modificadas y actualizadas frecuentemente, equipos de trabajos distribuidos geográficamente pueden trabajar de forma conjunta en proyectos de desarrollo, los servicios se pueden obtener a través de diversas fuentes que eliminan las fronteras internacionales, los costes se reducen mediante el uso de la infraestructura de un solo proveedor en lugar de mantener múltiples instalaciones hardware que a menudo provocan duplicidad de funciones o problemas de incompatibilidad y los gastos generales también pueden verse disminuidos gracias a la unificación de los esfuerzos de desarrollo.

Existen diferentes tipos de proveedores de PaaS, sin embargo, todos ellos ofrecen alojamiento y entornos de despliegue junto con varios servicios integrados. Las ofertas de servicios suelen variar en el nivel de escalabilidad y mantenimiento.

No existen demasiados trabajos que centren sus esfuerzos al nivel de la plataforma, pero es interesante como (Lawton, 2008) expresa la posibilidad de crear software basado en esta parte del paradigma de Cloud Computing debido a que, hablando en términos de configuración y personalización, el nivel PaaS permite a los diseñadores ser capaces de construir la plataforma cumpliendo las características y necesidades que sean consideradas en cada caso.

Es importante destacar que cualquiera de los modelos de prestación de servicio puede existir en un entorno aislado, de hecho, es imperativo comprender la estrecha relación que se establece entre ellos (Keller & Rexford, 2010). IaaS es la base de todos los servicios en la nube y se encuentra centrada en el nivel de arquitectura de la red. PaaS se encuentra articulado en IaaS, estableciéndose en el nivel de desarrollador de aplicaciones. Finalmente SaaS, se encuentra también articulado en IaaS y aparece como front-end para los usuarios finales.

2.3.2.2.1 Tipos de PaaS

A pesar de que los trabajos a nivel de plataforma no están muy extendidos, se ha establecido una clasificación de los tipos de PaaS que se pueden encontrar:

- **De capacidad de desarrollo adicional**. Estas capacidades permiten la personalización de aplicaciones SaaS existentes y, en algunos aspectos, son equivalentes a macro lenguajes de personalización de capacidades proporcionados con aplicaciones software como Lotus Notes o Microsoft Word. A menudo esto obliga a desarrolladores PaaS y sus usuarios a realizar suscripciones a la propia aplicación SaaS.
- **Entornos para desarrollo de programas autónomos**. Estos entornos no incluyen dependencias a nivel técnico, financiero o de licencias en aplicaciones SaaS específicas o servicios web y tratan de proporcionar entornos generalizados de desarrollo.
- **Entornos para entrega de aplicaciones**. Estos entornos no incluyen capacidad de desarrollo, depuración y pruebas como parte del servicio, deben ser proporcionados off-line. Los servicios ofrecidos generalmente se centran en seguridad y en escalabilidad bajo demanda.
- **Plataforma Abierta como Servicio**. Este tipo de PaaS no incluye alojamiento como tal ya que proporciona un software de código abierto que permite a los proveedores de PaaS ejecutar aplicaciones. Por ejemplo, AppScale permite a los usuarios desplegar aplicaciones desarrolladas para Google App Engine en sus propios servidores, proporcionando acceso y almacenamiento de datos desde una base de datos estándar SQL o NoSQL. Algunas plataformas abiertas permiten al desarrollador el uso de cualquier lenguaje de programación, cualquier base de datos, sistema operativo, servidor, etc. para desplegar sus aplicaciones (Chohan et al., 2010).

2.3.2.2.2 Características clave

A continuación se recogen las características más importantes que rodean a los entornos de Plataforma como Servicio:

Arquitectura Multitenant.

Una plataforma multitenant es aquella que emplea recursos de computación comunes, incluyendo hardware, sistema operativo, software y una única base de datos subyacente con un esquema compartido para soportar múltiples usuarios de manera simultánea. Este hecho se encuentra en contraste con la arquitectura tradicional cliente-servidor, que requiere un conjunto completo de hardware y software dedicado para cada tenant (cliente o usuario).

El multitenancy (o multipropiedad) se beneficia enormemente del concepto llamado "innovación colectiva". Cuando un número muy elevado de negocios/clientes utilizan exactamente la misma infraestructura operacional, todos se benefician de cada una de las

diferentes maneras en las que ponen a prueba la infraestructura compartida. Todos ellos tienen acceso a la funcionalidad más avanzada que se introduce por orden de una minoría innovadora. Todos ellos se benefician de la robustez de la estructura después de que cualquiera de ellos detecte una nueva amenaza o problema.

La fuerza del multitenancy reside en que cada uno de los propietarios individuales se encuentra en continua evolución. Esta situación varía mucho con respecto a single-tenancy, donde se limita la evolución a aquellos cambios que son percibidos y benefician a ese propietario individual. Realmente, el multitenancy marca la diferencia entre una aplicación SaaS que está destinada a desaparecer por obsolescencia y otra que continua evolucionando con la nube y ampliando sus posibilidades a través de una red abierta y conectada.

Interfaz de usuario personalizable

Una oferta PaaS debe proporcionar la capacidad para construir interfaces de usuario altamente flexibles a través del modelo "drag and drop", metodología que permite la creación y configuración de componentes del interfaz de usuario según sea preciso. Para comenzar con la construcción de las interfaces de usuario, deben estar disponibles algunos componentes estándar predefinidos de forma que puedan ser ensamblados con el mínimo código posible. Para personalizaciones más avanzadas, que pueden ser necesarias para cumplir requisitos de usuario específicos, debe existir una opción para que fácilmente se pueda invocar una librería o una opción más sofisticada para reutilizar componentes.

Más allá, dado el crecimiento de los dispositivos web, debe proporcionarse flexibilidad adicional para el uso de otras tecnologías como CSS, AJAX y Adobe Flex que permitan especificar la apariencia de la interfaz de las aplicaciones. Además, la oferta de la plataforma debe permitir a los diseñadores y desarrolladores un control completo sobre la forma de presentación de los datos dependiendo del contexto, es decir, mostrar la información de una determinada manera si se está accediendo a través de un dispositivo móvil y de otra si está ante un navegador de escritorio.

Este paradigma de construcción de interfaces de usuario basado en el modelo "drag and drop", propiedad esencial de plataformas PaaS robustas, proporciona a los diseñadores un mayor control sobre la apariencia de las aplicaciones y permite la creación de nuevas y sofisticadas capas de presentación de manera rápida y fácil sin precisar de grandes cantidades de código específico.

Personalización ilimitada de la Base de Datos

La persistencia de datos es un aspecto clave en muchas aplicaciones. Facilitar la creación, configuración y despliegue de objetos persistentes sin precisar conocimientos profundos de programación es una característica que debe cumplir una plataforma cloud potente. La plataforma PaaS debe soportar construcción de objetos, definición de relaciones entre esos

objetos y la configuración del comportamiento avanzado de los datos, todo a través de la sencillez de un navegador web.

De manera contraria a las bases de datos relacionales donde las tablas se utilizan para almacenar datos, los objetos constituyen los bloques fundamentales para las aplicaciones basadas en cloud. A través de una interfaz web declarativa que proporciona un completo control visual a nivel de metadatos, los diseñadores/desarrolladores de aplicaciones deben ser capaces de definir objetos con los campos y atributos que determinen qué tipo de datos se encuentra almacenado en cada entrada de objeto. La especificación de las relaciones entre objetos debe ser posible también a través de una interfaz web. Otra de las funciones obligatorias que se deben incluir es la de la habilidad para incorporar reglas de validación y permisos a nivel de objeto o campo.

Motor robusto de flujo de trabajo

La ejecución adecuada de procesos de negocio a través de la automatización es el primer objetivo que persiguen las aplicaciones de negocio. Una plataforma de computación en nube debe ofrecer un motor lógico de negocio que soporte la definición de flujos de trabajo y la especificación de reglas para engendrar la automatización de procesos.

Un proceso de flujo de trabajo define los diferentes estados de un objeto de negocio que fluye a través de su ciclo de vida. Las acciones de estos flujos de trabajo dirigen al objeto a través de diferentes estados dentro del contexto del proceso y puede ser desencadenado mediante la intervención humana o un evento.

Empleando una combinación de los procesos de flujos de trabajo, estados, acciones, eventos y las reglas que gobiernan las acciones, el diseñador de la aplicación debe ser capaz de modelar cualquier tipo de proceso de negocio empleando herramientas sencillas a través de un navegador. Además, la oferta PaaS debe incluir la capacidad de definir programáticamente potentes condiciones de disparador utilizando lenguajes de script.

Control granular sobre la seguridad (modelo de permisos)

Las plataformas PaaS deben proporcionar un sistema flexible de control que permita un control detallado sobre qué usuarios de las aplicaciones SaaS pueden ver y los datos a los que puede acceder cada usuario. Debe ser posible la definición desde acceso a nivel de aplicación (incluyendo menús, objetos, tablas, vistas, acciones, etc.) al campo individual. La definición de un modelo de control de acceso será posible gracias a la creación de grupos y roles y la asignación de los usuarios a estos grupos o roles.

Para implementaciones complejas a gran escala, debe disponerse de la capacidad para definir a qué características y datos puede acceder cada usuario. De esta manera los usuarios pueden ser segmentados a lo largo de la estructura organizacional para proveer de un acceso limitado a los datos y a las características de las aplicaciones.

Modelo flexible de integración de servicios

Los PaaS facilitan la rápida construcción de aplicaciones en el cloud mediante la proporción de los elementos base o fundamentos, como la persistencia de datos y la capacidad de manejar flujos de trabajo. Aún así, dados los complejos entornos de TI que penetran en las empresas en la actualidad, la oferta de los PaaS debe hacer uso de los principios de las SOA (Service Oriented Architectures) para permitir la continua integración de los datos de aplicación y la funcionalidad residente en la plataforma cloud con otros sistemas bajo demanda y sus aplicaciones.

Como mínimo, la oferta debe soportar un modelo de integración flexible habilitado a través de las APIs de SOAP y REST. Las respectivas APIs deben proporcionar métodos estándar para la creación, lectura, actualización y borrado, así como búsqueda, subida y bajada de archivos binarios, formas de trabajar con las relaciones y un método para recuperar completamente una representación XML de la definición de un objeto y todos sus componentes. Las APIs se deben adherir a las mismas restricciones de control de accesos y permisos que se especifiquen a través del modelo de seguridad.

Más allá, para acelerar la integración entre las aplicaciones cloud y los sistemas instalados en cliente, la plataforma cloud debe proporcionar un rango de conectores predefinidos y pre-construidos.

2.3.2.2.3 Ventajas de PaaS

De manera general se pueden establecer las siguientes ventajas que proporciona el modelo de Plataforma-como-Serivicio:

- Cada componente de la plataforma se proporciona como servicio (Middleware-como-Servicio, Comunicación -como-Servicio, Integración-como-Servicio, etc.)
- Proporciona los servicios requeridos para soportar el ciclo de vida completo de construcción y entrega de aplicaciones y servicios a través de Internet.
- Proporciona servicios para desplegar, probar, alojar y mantener aplicaciones de la misma manera que los entornos de desarrollo (IDE).
- El suministro de servicios incluye a múltiples usuarios utilizando el mismo entorno integrado de desarrollo de aplicaciones de manera concurrente.
- Tratándose de una propuesta de Cloud Computing, sigue un modelo de servicios de nube (pago por uso) donde no existe la necesidad de comprar un software, middleware o una licencia anual, sólo se paga en función de su uso.
- PaaS reduce el coste total de propiedad ya que no es necesario adquirir todo el sistema, programas, plataformas y herramientas necesarias para construir, ejecutar y desplegar aplicaciones. Los usuarios pueden alquilarlos por el periodo en el que serán utilizados.
- Incorpora escalabilidad y elasticidad para proporcionar la misma experiencia y eficiencia independientemente de la carga y el uso.

- PaaS es una opción perfecta para el desarrollo de metodologías ágiles.
- PaaS ayuda a la rápida construcción de aplicaciones en la nube proporcionando los elementos necesarios como servicios de flujos de trabajo que son esenciales para la creación de aplicaciones de negocio.
- Con PaaS, las características de los sistemas operativos pueden modificarse y actualizarse frecuentemente. Equipos de desarrollo distribuidos geográficamente pueden trabajar conjuntamente en proyectos de desarrollo software.
- Reducción de costes:
    - Gastos generales de servidores y almacenamiento. Cuando las fases de desarrollo y despliegue finalizan, los servidores y dispositivos de almacenamiento de la empresa permanecen inactivos, sin embargo requieren de potencia, enfriamiento y mantenimiento que incurren en una serie de gastos. Con PaaS las compañías evitan este problema ya que todas las herramientas de desarrollo son proporcionadas por el proveedor y no todas las iteraciones precisan de ser almacenadas en centros de datos.
    - Ancho de banda. Los procesos de desarrollo y despliegue pueden someter a estrés a la red junto con el centro de datos. Los equipos de desarrollo deben realizar pruebas exigentes para comprobar el comportamiento de las aplicaciones bajo diferentes circunstancias. Esta situación puede hacer disminuir la eficiencia de la red y surgir la necesidad de adquirir un mayor ancho de banda. PaaS habilita que las pruebas puedan realizarse en la nube en lugar de en centros de datos, lo que soluciona este problema.
    - Mantenimiento de software. El coste de gestión y actualización de software supone a menudo una carga y un gran coste para las organizaciones. Aunque el coste de la estructura PaaS requiere de un cobro “por usuario”, “por mes”, el coste se compensa con la reducción o eliminación de los costes de las licencias y las tasas anuales de mantenimiento. El proveedor del servicio gestiona las actualizaciones de hardware y software y proporciona seguridad física y software para la automatización de las tareas diarias.
    - Personal de soporte. Mediante la adopción de una plataforma estandarizada para toda la organización, los conflictos hardware y software se reducen considerablemente, simplificando el servicio y soporte. El nivel de estandarización y automatización permite a las organizaciones reducir o redirigir sus equipos fuera de estas tareas, con el consiguiente ahorro de costes.
    - Errores humanos. Durante los procesos de desarrollo y despliegue se expone a los profesionales a grandes presiones que en ocasiones desembocan en acciones descuidadas. Con PaaS se reduce el número de estos errores casi por completo ya que la plataforma ha sido probada completamente y se sabe cómo funciona.
    - Requiere mínimos conocimientos. Las herramientas de desarrollo a menudo resultan complejas y no se encuentran estandarizadas por lo que requieren de un alto grado de conocimientos y experiencia. La curva de aprendizaje de estas habilidades es inclinada y existe una necesidad de continua gestión de esos componentes. Mediante la proporción de las herramientas y el middleware,

las plataformas PaaS reducen el nivel de conocimientos requerido para desplegar aplicaciones y elimina el cuello de botella que puede generarse al precisar de la asistencia de un especialista.

- PaaS supone un incremento de la velocidad, flexibilidad y agilidad a los procesos de desarrollo. Mediante la proporción de una infraestructura de aplicación previsible y heterogénea, las organización no se sobrecargan habilitando aplicaciones y pueden responder con rapidez a las necesidades de sus clientes. PaaS contribuye a lo siguiente:
    - Permite reducir los tiempos de entrada al mercado permitiendo a los equipos de desarrollo centrarse en la aplicación.
    - Mejora la reacción a los cambios y oportunidades debido a que la organización no posee los grandes costes anticipados de las aplicaciones tradicionales.
    - Permite extender las inversiones de capital lo que hace que la compañía sea más competitiva.

2.3.2.3 PaaS vs IaaS

A pesar de que tanto SaaS como IaaS incrementan la eficiencia operacional, son completamente diferentes (L.M. Vaquero, 2011).

En los modelos IaaS, el proveedor sólo proporciona la infraestructura subyacente, incluyendo red, almacenamiento, recursos computacionales y tecnologías de virtualización. En este modelo, los desarrolladores aún precisas de un gran trabajo operacional. Deben conseguir un sistema operativo, configurarlo manualmente y gestionar y actualizar numerosos componentes.

En el modelo PaaS, el proveedor no sólo proporciona los recursos de la infraestructura subyacente sino también la plataforma para el desarrollo de aplicaciones. Esta plataforma incluye automatización de despliegue, pruebas e iteración de aplicaciones. El sistema operativo, bases de datos, middleware y las herramientas y servicios actualizados son proporcionados por el suministrador PaaS. Tareas que conllevan tiempo como las de configuración, optimización y actualización constante del entorno son realizadas por el proveedor, pero el cliente continúa manteniendo el control total sobre su entorno.

Al comparar ambos modelos se puede decir que PaaS es más sencillo, barato y rápido de desplegar y escalar.

2.3.3 **Modalidades de despliegue y consumo**

Independientemente de los modelos de prestación, se pueden distinguir cuatro modalidades de despliegue y consumo de los de servicios de computación en nube (Armbrust et al., 2010). A continuación se presenta una pequeña descripción de cada una de ellas.

2.3.3.1 Nubes privadas

Las nubes privadas son propiedad de una organización o su correspondiente proveedor de servicios. Ofrecen un entorno operacional dedicado (single-tenant) con todos los beneficios, funcionalidad, escalabilidad y elasticidad propios del Cloud Computing.

La infraestructura es propiedad y/o está físicamente localizada en una organización o su correspondiente proveedor de servicios, quien se encarga de la gestión de la nube y los controles de seguridad.

Los consumidores de los servicios de las nubes privadas se consideran "de confianza" ("trusted") y son normalmente parte de la estructura legal y contractual de la organización (empleados, proveedores, socios comerciales, etc.). Los consumidores que no se consideran fiables son todos aquellos que pueden acceder a todos o algunos de los servicios pero que no forman parte de la organización.

2.3.3.2 Nubes públicas

Este tipo de nubes son propiedad de los proveedores de servicios externos y pueden ofrecer un entorno operativo tanto dedicado (single-tenant) como compartido (multitenant) con todos los beneficios, funcionalidad, escalabilidad y elasticidad propios del Cloud Computing.

Las infraestructuras de las nubes públicas son también propiedad de dichos proveedores de servicios y se ubican físicamente en sus centros. Asimismo, la responsabilidad de su gestión y mantenimiento recae en estos proveedores.

Todos los consumidores de las nubes públicas se consideran no fiables.

2.3.3.3 Nubes gestionadas

Este tipo de nubes son proporcionados por los proveedores de servicios y pueden ofrecer un entorno operativo tanto dedicado (single-tenant) como compartido (multitenant) con todos los beneficios, funcionalidad, escalabilidad y elasticidad propios del Cloud Computing.

Las infraestructuras físicas son propiedad de la organización y se localizan físicamente en sus centros de datos. No obstante, la gestión y control de las mismas se deja en manos de los proveedores de servicios.

Los consumidores de dichas nubes pueden ser tanto fiables como no fiables.

2.3.3.4 Nubes híbridas

Las nubes híbridas combinan las características da las nubes públicas y las privadas, ofreciendo intercambio de información y posible portabilidad y compatibilidad de aplicaciones entre diferentes nubes, haciendo uso de metodologías estandarizadas o propietarias independientemente del propietario o localización.

Los consumidores de dichas nubes pueden ser tanto fiables como no fiables.

2.3.4 **Debilidades del Cloud Computing**

A pesar de las expectativas que genera y de las oportunidades que ofrece, Cloud Computing sigue teniendo que cumplir con los requisitos usuales de seguridad, privacidad, disponibilidad, etc., propios de todas las tecnologías de información. Asimismo, surgen nuevas preocupaciones relacionadas con el hecho de que las empresas confían a la nube la gestión de su información estratégica de negocio.

2.3.4.1 Seguridad

Las empresas ya no guardan sus datos en infraestructuras internas a la propia empresa, sino que deben delegar su almacenamiento a proveedores de la nube. Eso implica que dichos proveedores deben asegurar la privacidad, consistencia y seguridad de los datos frente a amenazas externas y de corrupción. Por tanto, no sólo se deben seguir las mejores prácticas de seguridad y encriptación, sino también se hace imperativo reforzar la transparencia de las operaciones.

El diseño de aplicaciones para la nube tiene que tener en cuenta las nuevas características del Cloud Computing, por lo que se deben actualizar todas las prácticas y estándares vigentes actualmente. Las aplicaciones que se ejecutan en la nube presentan diversos riesgos en función del modelo de prestación de servicio y modalidades de consumo. A continuación se comentarán algunos puntos a tener en cuenta.

2.3.4.1.1 IaaS

En las plataformas de Infraestructura con servicio, el proveedor ofrece un conjunto de componentes virtualizados, como son la máquina virtual, almacenamiento u otros componentes que pueden servir para desarrollar y ejecutar una aplicación. En entornos IaaS, los datos almacenados en la máquina virtual no se hacen persistentes cuando la máquina se apaga, por lo que el proveedor normalmente proporciona almacenamiento remoto.

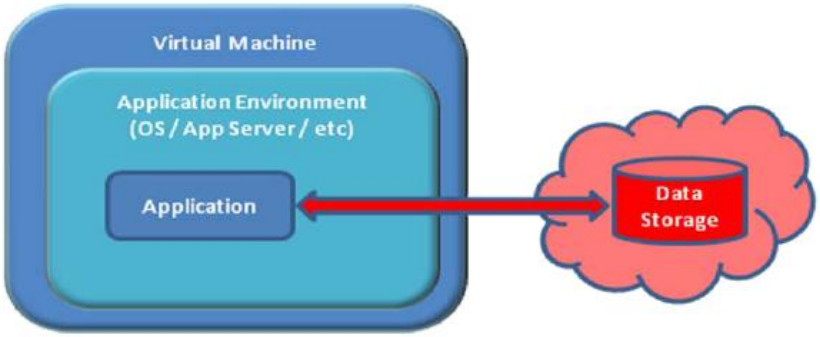


Figura 10. Almacenamiento remoto en IaaS

La arquitectura para las aplicaciones IaaS es muy parecida a la de las aplicaciones web tradicionales, es decir, arquitecturas distribuidas de n capas. Para las aplicaciones que se ejecutan dentro de los límites de una empresa, existen numerosos controles físicos y lógicos para garantizar la seguridad de los hosts y las redes de la que hacen uso dichas aplicaciones. En el caso de la computación en nube, dichas medidas son inexistentes y deben reemplazarse por otras a nivel de aplicación.

Los proveedores de servicios IaaS ponen a disposición de los clientes una gran cantidad de imágenes de máquinas virtuales, algunas de ellas ofrecidas por los propios proveedores y otras por otros clientes. Cuando una empresa usa una máquina virtual, ésta debe cumplir con las mismas políticas y prácticas de seguridad que las establecidas dentro de la empresa. Una posible solución sería que la propia empresa cliente se encargue de configurar su propia imagen de máquina virtual dentro del entorno IaaS del proveedor, asegurando que cumpla los requisitos exigidos (Luis M. Vaquero, Rodero-Merino, & Morán, 2011).

Una buena práctica de seguridad para las aplicaciones basadas en cloud es construir imágenes de entorno operacional y de aplicación personalizadas, que únicamente tengan las capacidades necesarias para soportar la pila de la aplicación. Limitando la capacidad de la pila al mínimo exigible por la aplicación se reduce tanto la "superficie" del host vulnerable a ataques externos como los parches sucesivos necesarios para mantener dicha pila segura.

La mayoría de las aplicaciones distribuidas de una empresa no toman ninguna medida adicional para asegurar la seguridad de las comunicaciones siempre y cuando éstas no transgredan las fronteras de la red de la propia empresa. En el caso del Cloud Computing, es la aplicación la responsable de garantizar la seguridad de las comunicaciones ya que se ejecutan en un entorno compartido con otros clientes. Los mecanismos de seguridad pueden variar en función del tipo de aplicación. En el caso de comunicaciones síncronas, como conexiones punto a punto, asegurar el canal de comunicación sería suficiente. Sin embargo, en el caso de comunicaciones asíncronas, como el paso de mensajes, los mecanismos de seguridad deben proteger la integridad del mensaje (Hay, Nance, & Bishop, 2011).

Las plataformas IaaS hacen uso de unas claves secretas para identificar cuentas de acceso válidas. Dichas claves deben figurar en todas las llamadas a los servicios del proveedor de la nube, por lo que las empresas deben tomar medidas oportunas para proteger dichas claves.

En resumen, en lo referente a la información confidencial, las aplicaciones deben pasar los controles de seguridad internos de la empresa, ampliándolos para adaptarse a los riesgos inherentes a la computación en nube cuando sea oportuno.

2.3.4.1.2 PaaS

El proveedor de PaaS ofrece no solamente un entorno de ejecución para la aplicación, sino también una capa adicional de integración y middleware. Por ejemplo, un entorno IaaS proporciona una cola de mensajes para las comunicaciones asíncronas, mientras que PaaS añade además un Enterprise Service Bus (ESB) que ofrece tanto la cola de mensajes como algunos servicios relacionados, como puede ser el enrutamiento de mensajes. En la

Figura 11 se muestran las capas relevantes del modelo PaaS:

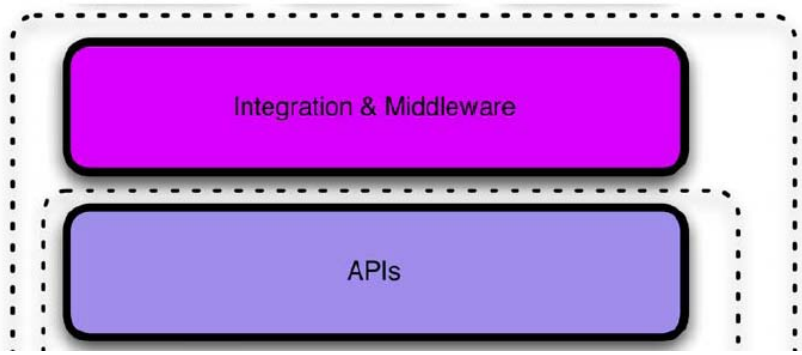


Figura 11. Capas de PaaS (www.rationalsurvivability.com)

La plataforma PaaS proporciona también un entorno de programación para poder acceder y utilizar los servicios de la capa de integración y middleware. La inclusión de ese entorno adicional afecta en gran medida al diseño y ejecución de las aplicaciones. Por ejemplo, el entorno de programación puede limitar los servicios del sistema operativo a los que una aplicación puede tener acceso, con el fin de mejorar la eficiencia en entornos compartidos (multitenant).

El impacto en la seguridad en plataformas PaaS es similar al descrito en el apartado anterior. Aunque PaaS proporciona un conjunto de servicios adicionales sobre IaaS, estos servicios siguen siendo compartidos por todos los usuarios de la nube. En consecuencia, se deja en manos de la aplicación el asegurar las integridades de las comunicaciones y de la información confidencial, incluidas las claves de acceso.

2.3.4.1.3 SaaS

El modelo SaaS ofrece la gestión de las mismas infraestructuras y entorno de programación, ampliándolo con funcionalidades y aplicaciones específicas. Dichas funcionalidades pueden tanto ofrecer servicios determinados al usuario final, como pasar a ser un componente dentro de la aplicación de usuario. Las funcionalidades también pueden ampliarse por los clientes mediante su propio código personalizado. Las aplicaciones externas pueden comunicarse con aplicaciones SaaS mediante la API proporcionada por los proveedores de la plataforma, tal y como se puede observar en la Figura 12:

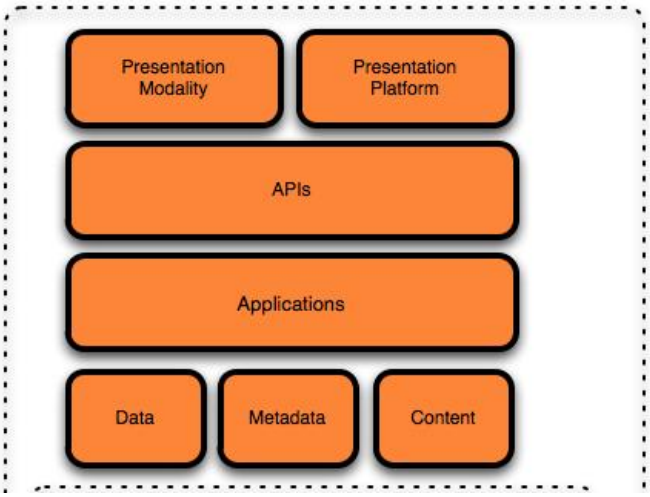


Figura 12. Estructura de plataforma SaaS (www.rationalsurvivability.com)

La plataforma SaaS hereda todas las características de las plataformas IaaS y PaaS y, por lo tanto, hereda también sus debilidades y soluciones. El nivel de seguridad para las ampliaciones personalizadas de funcionalidades y aplicaciones ofrecidas por SaaS debe ser el mismo que el de las funcionalidades o aplicaciones originales. Asimismo, la comunicación con entidades externas a través de las APIs debe cumplir con las normativas y estándares de seguridad propios de cualquier intercambio de datos externo.

Las plataformas SaaS afectan además al ciclo de vida del desarrollo software. Si bien esta afirmación es cierta para los tres modelos descritos, cobra especial relevancia en entornos SaaS, puesto que las aplicaciones son compartidas entre las empresas y los proveedores.

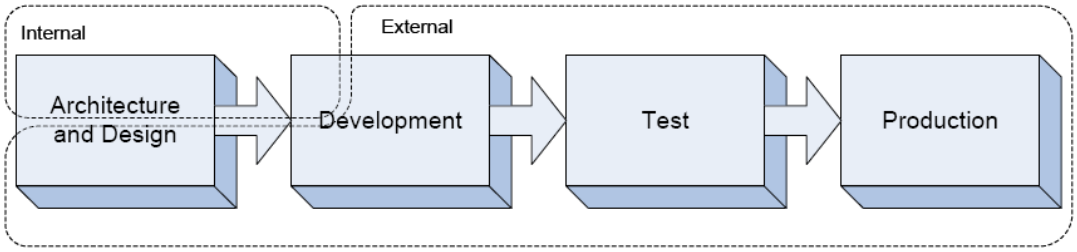


Figura 13. Ciclo de vida de desarrollo de software en SaaS

Por último, cabe destacar que las medidas concretas de seguridad a implantar en cada caso, tales como encriptación de los datos o control de acceso de los usuarios, están estrechamente relacionadas con las normativas legales vigentes, y comentadas en el apartado "Cumplimiento de normativas".

2.3.4.2 Disponibilidad y fiabilidad

Este problema está estrechamente relacionado con lo expuesto en la sección anterior. Ningún usuario desea entregar a los proveedores datos críticos (personales o de negocio) sin tener la confianza de poder disponer de los mismos en todo momento. Esto plantea la necesidad de considerar, entre otras cosas:

- **Replicación de los datos**. ¿Se mantiene una copia de los datos? ¿Dónde se guarda? ¿Cada cuánto tiempo se actualiza?
- **Recuperación frente a fallos**. ¿Qué pasa si ocurre una pérdida de datos? ¿Se puede recuperar la última versión? ¿En cuánto tiempo?
- **Disponibilidad total del sistema**. ¿Se puede caer la cloud? ¿Qué pasa si se cae?

Este problema abarca dos vertientes. Por un lado, los proveedores deben crear y seguir procedimientos eficaces que garanticen la disponibilidad de la información. Por otro lado, los usuarios, sobre todo los clientes empresariales, sienten mucha inquietud al llevar información crítica de negocio fuera de sus instalaciones, donde tienen un control total, y confiarla a terceros. Dichos clientes reclaman la necesidad de Acuerdos de Nivel de Servicio (SLA, del inglés Service Level Agreement) para fijar el nivel acordado para la calidad del servicio. Un SLA es un contrato escrito entre un proveedor de servicio y su cliente, con el objetivo de definir las necesidades del cliente a la vez que controlar sus expectativas de servicio en relación a la capacidad del proveedor, proporcionar un marco de entendimiento y reducir las áreas de conflicto. A día de hoy existen todavía una serie de obstáculos que deberán ser resueltos para que Cloud Computing se convierta en una opción globalmente aceptada (Kearney & Torelli, 2011).

En primer lugar, existe una falta de SLAs por parte de proveedores de la nube. Amazon, por ejemplo, ofrece SLAs para sus servicios de S3 (del inglés Simple Storage Service) y EC2 (del inglés Elastic Compute Cloud), que son sólo dos de sus once servicios ofrecidos en el paquete AWS (del inglés Amazon Web Services) (Garfinkel & Garfinkel, 2007).

En segundo lugar, en el caso de Cloud Computing, los SLAs pueden ser particularmente difíciles de definir, puesto que deben centrarse en el servicio que se está proporcionando más que en cómo se está haciendo. Las métricas usuales como ancho de banda o latencia pierden su relevancia. Los usuarios quieren saber si pueden acceder a un determinado servicio y en cuánto tiempo, no si el proveedor cumple con los valores de latencia establecidos. Aunque estos valores están relacionados, lo que importa al cliente es el resultado final, que depende de un conjunto de factores (rendimiento de la infraestructura, comportamiento de la red, etc.) y es, en consecuencia, más difícil de asegurar.

Al mismo tiempo, se eleva el nivel de exigencia por parte de los clientes a la hora de negociar los SLAs debido al aumento de la criticidad de los datos y aplicaciones traspasadas a la nube.

Por último, existe un debate acerca de si los SLAs son los mecanismos adecuados para generar confianza en los proveedores.

2.3.4.3 Monitorización

El seguimiento (monitorización), ya sea del rendimiento o disponibilidad, es de vital importancia en cualquier servicio de TI. Los clientes acusan la falta de herramientas proporcionadas por los proveedores para monitorizar lo comportamiento del sistema.

De nuevo, el problema surge a raíz del carácter innovador de Cloud Computing, que invalida las herramientas usadas en sistemas tradicionales. Por ejemplo, los parámetros usados normalmente para medir el rendimiento del sistema, como el uso de la CPU o memoria, no son representativos en la mayoría de los casos del rendimiento real cuando se trata de computación en la nube. Para las aplicaciones en cloud resulta más relevante fijarse en el rendimiento de las transacciones u operaciones de E/S a disco.

Además, y debido a la ya comentada preocupación de los clientes empresariales por la seguridad de la información, los usuarios demandan tener un control exhaustivo sobre sus datos. Si las instancias de un usuario se colocan en la nube junto con las instancias de otros, el usuario querrá saber si esas otras instancias afectan negativamente a la capacidad de procesamiento de sus aplicaciones. En otras palabras, las herramientas de seguimiento deben permitir monitorizar no sólo lo que hace el sistema de manera eficaz, sino también lo que ocurre en la red alrededor del sistema y quién la está usando. Esto complica aún más las cosas ya que los servicios de la nube están cambiando constantemente: se lanzan más servidores para gestionar más carga, se quitan máquinas cuando ya no hacen falta, etc.

Un factor más a tener en cuenta es proporcionar formatos consistentes para que las nuevas herramientas sean compatibles con los sistemas de monitorización existentes. Los proveedores han tomado conciencia de esta necesidad y se han empezado a desarrollar

herramientas de seguimiento que satisfagan las demandas de los usuarios. Algunos ejemplos son CloudStatus de Hyperic, Nagios o Nimsoft Monitoring Solutions.

2.3.4.4 Gestión e integración

Otro de los retos que se plantean a los servicios de la nube es su habilidad de integrarse en el ámbito tecnológico actual de la empresa y personalizarse sus características para adaptarse al contexto específico de negocio.

2.3.4.5 Cumplimiento de las normativas

Una de las preocupaciones más inmediatas con respecto al Cloud Computing es la falta de una legislación actualizada y globalmente aceptada que se adapte al panorama tecnológico contemporáneo. A continuación se describen brevemente los principales puntos de conflicto existentes.

2.3.4.5.1 Privacidad de los datos

El debate se centra en torno a la privacidad de la información que los usuarios, particulares y empresariales, deberán ceder a los gestores de la nube. Los clientes confían datos personales y de negocio a los propietarios de la cloud y quieren asegurarse de que el procesado de estos datos se hará según las normativas legales establecidas, que garanticen la privacidad e integridad de los mismos. Actualmente, existen diversas leyes, tanto a nivel nacional como de la Unión Europea, cuyo objetivo principal es regular y proteger el tratamiento de datos de carácter personal, y como consecuencia también la información empresarial, gestionada a través de Internet. Entre otras, cabe destacar la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal (LOPD), así como el Real Decreto 1720/2007, de 21 de diciembre, por el que se aprueba el reglamento de desarrollo de la LOPD (RDLOPD). Dichas normativas prevén una serie de medidas de seguridad, tanto técnicas como organizativas, con el objetivo de velar por la seguridad de los datos, cuyo incumplimiento podrá acarrear una serie de sanciones preestablecidas. Algunos ejemplos de estas medidas son:

- Establecimiento de sistemas de identificación y autenticación de usuarios
- Definición de procesos de realización de copias de respaldo (backups)
- Establecimiento de cifrado de información cuando se transmita por red pública etc.

A nivel europeo, se debe mencionar la Directiva de Protección de Datos (Directiva 95/46/EC), que regula el procesamiento de datos personales en el ámbito de la Unión Europea, siendo un componente importante dentro de las leyes de privacidad y derechos humanos.

2.3.4.5.2 Auditoría

Una manera de que los clientes evalúen si el proveedor de la nube cumple con los requisitos legales es mediante la auditoría, que ayuda a esclarecer algunas de las operaciones internas de dicho proveedor. Sin embargo, aquí se presenta un doble problema. Y es que, en primer lugar, el mismo concepto de auditoría va en contra del principio fundamental del Cloud Computing, el cual intenta abstraer los detalles de bajo nivel ofreciendo APIs e interfaces externas fáciles de usar. En segundo lugar, los auditores deben afrontar el reto de entender verdaderamente el concepto de la Computación en Nube para así poder mejorar los procedimientos de auditoría actuales con el fin de asegurar que los mismos permanezcan vigentes dentro del nuevo contexto tecnológico.

Para que los clientes confíen sus datos a la nube, deben asegurarse que dicha nube cumple al menos con las mismas normativas que se ven obligados a cumplir los clientes según la naturaleza de su negocio. Por ejemplo, no van a exigir la misma seguridad los clientes particulares que quieran guardar algunos de sus archivos en la nube, que una empresa de venta por internet que acepte pagos mediante tarjetas o un hospital. Algunas de las normativas o estándares que los proveedores de nubes pueden verse obligados a cumplir son:

- ISA/IEC 27001 es el estándar para la seguridad de la información, aprobado y publicado como estándar internacional en Octubre de 2005 por International Organization for Standardization y por la comisión International Electrotechnical Commission. Especifica los requisitos necesarios para establecer, implantar, mantener y mejorar un Sistema de Gestión de la Seguridad de la Información (SGSI).
- SAS 70 (del inglés Statement on Auditing Standards No. 70: service Organizations). Es un estándar aprobado por Auditing Standards Board de American Institute of Certified Public Accounts (AICPA) que define los estándares profesionales usados por un auditor para evaluar los controles internos de los servicios de una organización. El proveedor de la nube debe ser capaz de describir lo que está pasando, dónde la información entra en la nube, qué se hace cuando el proveedor la recibe, cómo vuele a los usuarios, los controles sobre el procesamiento de datos y, lo más importante, qué se hace con los datos que entran en la nube. En otras palabras, el cumplimiento de SAS 70, se basa en el hecho de que hay información numérica concreta proveniente de los datos que han sido almacenados o tratados dentro de la cloud. La principal ventaja del SAS 70 consiste en facilitar la construcción de un entorno de control que los auditores consideren apropiado.
- PCI DSS (del inglés Payment Card Industry Data Security Standard) y significa Estándar de Seguridad de Datos para la Industria de Tarjetas de Pago. El cumplimiento de la normativa PCI DSS se ve complicado por el hecho de que parte del procesamiento de las transacciones realizadas con dichas tarjetas debe realizarse en el punto de venta del proveedor, aunque el resto de procesamiento se haga en la nube. Entre las responsabilidades de los proveedores, según esta normativa, se pueden citar los firewalls, detección de intrusiones, recuperación ante fallos, controles físicos, etc. La

parte del cliente también tiene que cumplir ciertos requisitos de seguridad y, evidentemente, resulta mucho más difícil de controlar.

- HIPAA (del inglés Health Insurance Portability and Accountability Act), se refiere a ley de Portabilidad y responsabilidad del seguro Médico, aprobada por el Congreso el 21 de Agosto de 1996. El proveedor de la nube debe ser capaz de asegurar a aquellos de sus clientes que requieren un entorno que cumple con HIPAA (por ejemplo, los encargados de la atención médica civil y militar, proveedores médicos etc.) de que su nube es uno de esos entornos. Para ello, debe asegurarse de que la nube es adecuada para garantizar la seguridad tanto física como lógica de la información, según las reglas de seguridad establecidas por HIPAA. Por ejemplo, el cifrado de información no es un requisito imprescindible de HIPAA, pero en caso de que la información no esté cifrada, se requieren ciertos mecanismos de control para prevenir accesos no autorizados. Sin embargo, sí se exige que los datos enviados a través de una red pública estén cifrados.

2.3.4.5.3 Movimiento internacional de datos

Otro problema surge a raíz de la propia naturaleza de los servicios del Cloud Computing. Los proveedores ofrecen almacenamiento de datos en sus servidores, repartidos por todo el mundo. Este servicio se realiza de manera transparente al usuario: los datos de fragmentan, se replican y se almacenan en los servidores elegidos por el proveedor con tal propósito, pudiendo cambiar dinámicamente para adaptarse a la demanda del usuario, pero sin que éste tenga control sobre el proceso. Este enfoque aporta la ventaja de la abstracción de los servicios de bajo nivel. No obstante, obliga al proveedor a cumplir no sólo las normativas legales locales, sino también las internacionales, ya que es muy probable que muchas de las aplicaciones web acaben en servidores extranjeros.

Según Leopoldo Mallo, director general de LOPDGEST "El Movimiento Internacional de Datos, viene regulado en la normativa en materia de Protección de Datos, concretamente en la Instrucción 1/2000, de 1 de diciembre de la Agencia Española de Protección de Datos, en la cual se prohíbe como regla general la Transferencia Internacional de Datos con destino a países que no proporcionen un nivel de protección equiparable al establecido en la LOPD, sin la previa autorización del Director de la Agencia Española de Protección de Datos, salvo determinadas excepciones previstas en la normativa de referencia. Así mismo, también se recoge la obligación de cumplimiento de una serie de requisitos tales como, el deber de información y la correspondiente notificación de la Transferencia a la Agencia Española de Protección de Datos". Los países que se consideran seguros son todos los de la Unión Europea, Argentina, Islandia, Liechtenstein, Noruega y Suiza, pues proporcionan un nivel de protección equiparable a la legislación europea. En cuanto a los Estados Unidos y Canadá, la Comisión Europea, mediante su Decisión de 26 de Julio de 2000, y con arreglo a la Directiva 95/46/CE del Parlamento Europeo y del Consejo, considera que tienen un nivel de protección adecuado aquellas entidades que estén adscritas a los "principios de puerto seguro" o sujetas al ámbito de aplicación de la ley canadiense de protección de datos, respectivamente.

2.3.4.5.4 Propiedad intelectual

Las responsabilidades jurídicas y la propiedad intelectual son algunas de las cuestiones a considerar cuando se habla del Cloud Computing. Para algunos servicios, la solución es trivial: el propietario de la nube posee la infraestructura y las aplicaciones, mientras que los usuarios son dueños de sus datos y resultados de cualquier operación o tratamiento de los mismos. Sin embargo en el caso de mashups (aplicaciones híbridas que usan los resultados de otras aplicaciones)(Benslimane, Dustdar, & Sheth, 2008) o SaaS (componentes software como servicio) las fronteras se hacen más borrosas y no se ve tan claro quién posee qué, hasta qué punto y cuáles son los derechos y responsabilidades de los proveedores y clientes de la nube. Un asunto adicional a considerar es el de fin de servicio: qué sucede con los datos (y todas sus copias) y las aplicaciones de los clientes una vez que su relación con el proveedor se termina.

2.3.4.5.5 Interoperabilidad y portabilidad

Con la creciente necesidad de flexibilidad, disponibilidad y alto rendimiento, se aprecia una demanda cada vez mayor de estándares universales que permitan interoperabilidad y portabilidades de las aplicaciones entre diferentes estructuras de computación en nube, algo que todavía no existe.

Interoperabilidad

Se puede definir la interoperabilidad como la capacidad de dos sistemas de intercambiar información de manera que el receptor sea capaz de entenderla y, por consiguiente, hacer uso de ella.

Existen varias iniciativas actualmente orientadas a mejorar ese aspecto del Cloud Computing. Una de ellas es la estandarización de la interfaz de nubes públicas, promovida por el OGF Open Cloud Computing Interface Working Group (OCCI-WG). Este grupo está trabajando para crear una interfaz estándar unificada para la gestión de la infraestructura del Cloud Computing. Dicha interfaz permitirá el desarrollo de diversas herramientas para controlar las tareas comunes, que sean capaces de interoperar y colaborar entre sí. El grupo de trabajo mantiene una lista completa de todas las APIs de nubes existentes y una serie de comparativas entre ellas.

Portabilidad

La portabilidad hace referencia a la facilidad con la que un componente puede ser transferido de una nube a otra, sin que ello afecte a su correcto funcionamiento. El DMTF Open Virtualization Format (OVF) proporciona un esquema para describir una máquina virtual o una colección de ellas. Pretende ser un medio para los usuarios de expresar sus necesidades de infraestructura a los proveedores IaaS de manera estandarizada. Este formato es un primer

paso hacia la portabilidad de máquinas virtuales entre proveedores de nubes ya que permite describir las características deseadas de una máquina en un lenguaje unificado. Esta descripción luego puede traducirse a una representación interna propia de la infraestructura de cada proveedor.

No obstante, definir una interfaz estandarizada es solamente una parte de la solución al problema de portabilidad, pues únicamente garantiza que una aplicación puede funcionar correctamente en diferentes entornos. Pero, ¿y qué pasa con el rendimiento? Pues no es suficiente con que la aplicación funcione bien, debe ser igual de eficiente. Sin embargo, ésta es una cuestión mucho más compleja debido a las diferentes políticas de asignación de recursos, auto-configuración y restricciones a la hora de desplegar máquinas virtuales en las nubes de los proveedores.

Por último, cabe mencionar que los esfuerzos por mejorar la portabilidad en el Cloud Computing ayudan a mitigar la preocupación existente de dependencia del proveedor. Los usuarios temen que una vez desarrolladas las aplicaciones para una plataforma, la dificultad para cambiar a otra les obligará a usar siempre los servicios del mismo proveedor. Una mayor portabilidad les otorga más libertad a los usuarios a la hora de elegir la nube en la que quieren desplegar y ejecutar sus aplicaciones.

2.3.5 Ventajas de SaaS y Cloud Computing

Como se ha explicado a lo largo de toda la sección 2.3, el Software como servicio se refiere a un modelo de distribución de software donde el suministrador del mismo aporta, además del mismo, servicios tales como mantenimiento, ayuda y soporte. La ventaja que ofrece es que el software requerido se distribuye y aloja en Internet, por lo que el usuario no necesita en ningún momento instalarlo en su computadora personal, y todo lo referente a seguridad, calidad de servicio o rendimiento queda a expensas del proveedor.

La gran ventaja de este modelo es su alta escalabilidad, reducción drástica de costes y eficiencia máxima en términos de rendimiento y de disponibilidad para los Sistemas de Información Corporativos para cualquier segmento de mercado, ya sea una corporación grande, un usuario estándar o una pequeña empresa.

El SaaS y la computación en nube presentan además conjuntamente las siguientes ventajas:

- No requiere, por parte del cliente, la posesión de un área o infraestructura especializada para soportar el sistema, lo que infiere en una **disminución de los costes y riesgos**.
- Al tratarse de un servicio contratado por el cliente a la compañía suministradora del software, ésta ofrece la **garantía de operatividad** de la misma, es decir, el aval de que la aplicación estará disponible en todo momento y que funcionará de manera correcta a lo acordado en los términos del acuerdo suscrito con el comprador.
- La responsabilidad de la operación recae en la empresa proveedora de servicios. Esto significa que la garantía de disponibilidad de la aplicación y su correcta funcionalidad, es parte del servicio que da la compañía proveedora del software.

- Asimismo, este contrato que se establece entre la empresa suministradora y el cliente, garantiza que el servicio y atención a este segundo será siempre **continuo y constante**, para que el último siga pagando por la aplicación.
- La empresa proveedora, suministra los medios seguros de acceso en los entornos de la aplicación. Si una empresa quiere dar opciones SaaS en su cartera de productos debe ofrecer accesos seguros para que no se infiltren datos privados en la red pública.
- **Reducción de costes**: los usuarios pueden reducir los costes fijos relacionados con el hardware, software y los propios servicios y, en su lugar, pagar únicamente por aquello que van a utilizar. Esto supone, a su vez, reducir la barrera de entrada (tecnológica) para pequeñas y medianas empresas (PYMES) que no pueden permitirse la adquisición de grandes equipos informáticos de última generación o pagar los exorbitados precios de las licencias de determinados productos software.
- **Independencia de dispositivo y localización**: los usuarios pueden acceder a los recursos y servicios disponibles "en la nube" desde cualquier dispositivo (p.ej. PC o teléfono móvil) desde cualquier lugar (a través de Internet).
- **Multi-propiedad**: facilita la compartición de recursos y costes entre un largo número de usuarios. Esto permite, por ejemplo, la centralización de la infraestructura en áreas con bajos costes (en propiedades, electricidad, etc.) y la mejora de la utilización y eficiencia de los sistemas informáticos (generalmente, infrautilizados).
- **Mejora de la confianza**: la redundancia de los sistemas en la computación en nube permite asegurar el correcto funcionamiento del entorno.
- **Escalabilidad**: la provisión de recursos o servicios bajo demanda y el uso de arquitecturas no acopladas permite mejorar la escalabilidad de los sistemas.
- **Mejora de la seguridad**: las compañías TIC proveedoras de los recursos en la computación en nube pueden permitirse, generalmente, la aplicación de sistemas de seguridad más avanzados que los disponibles en las compañías primordialmente consumidoras de productos software.
- **Sostenibilidad**: la mejora en la utilización de los recursos y la creación de sistemas más eficientes inciden en el ahorro energético y el desarrollo sostenible.

Otro valor añadido que aporta este modelo de distribución de software es que no requiere de la adquisición de una licencia para el uso de las aplicaciones, sino que esto se ve paliado con el pago por el alquiler para el uso del servicio

2.4 Web Semántica

El World Wide Web (WWW) o Internet fue inventado en 1989 por Tim Berners-Lee (Tim Berners-Lee et al., 1992) y este evento cambió totalmente la forma de gestionar y acceder a la información. Actualmente, la Web es un enorme repositorio en continuo crecimiento, en el que cualquiera puede publicar documentos y ponerlos a disposición del mundo, lo que ha supuesto una revolución en las aplicaciones y en el acceso a la información, así como ha servido para abrir puertas a nuevas formas de negocio. Sin embargo, el lenguaje utilizado para

la confección de estos documentos (o páginas Web) sólo fue diseñado para la lectura humana, ya que únicamente permite especificar cómo debe presentarse la información en la pantalla (color del texto, cabeceras, párrafos, etc.). Esto implica que cuando un usuario accede a una página de un buscador para localizar una determinada información, el ordenador no entiende el significado semántico de la búsqueda, sino que en todo caso realizará una búsqueda a nivel léxico de las palabras.

Como se comentaba, la Web ha cambiado profundamente la forma en la que nos comunicamos, hacemos negocios y realizamos nuestro trabajo. Es posible establecer una comunicación con el mundo a bajo coste y cada vez con menos limitaciones técnicas. Podemos realizar transacciones económicas, tenemos acceso a millones de recursos independientemente de nuestra situación geográfica e idioma, etc. Todos estos factores han contribuido al éxito de la Web. Sin embargo, ha surgido un gran cuello de botella cuando se trata de explotar la información existente, es decir, resulta un gran problema encontrar una parte específica y concreta de la información buscada. Se puede decir que los factores que han llevado al éxito de la Web, también han originado sus principales problemas: sobrecarga de información y heterogeneidad de fuentes de información con el consiguiente problema de interoperabilidad.

La Web Semántica (T. Berners-Lee et al., 2001; Shadbolt, Berners-Lee, & Hall, 2006) fue concebida con el propósito de solucionar estos problemas permitiendo a los usuarios delegar tareas en software. Su objetivo es el de añadir semántica a los datos publicados en la Web (es decir, establecer e indicar el significado de los datos), de manera que las máquinas sean capaces de procesar esos datos, razonar con ellos, combinarlos y realizar deducciones lógicas de manera muy similar a como lo haría un ser humano. Por ese motivo, se puede decir que la Web Semántica se caracteriza por la asociación de semánticas formales accesibles y entendibles por las máquinas y el contenido tradicional de la Web.

Según la W3C ''La Web Semántica es una Web extendida, dotada de mayor significado en la que cualquier usuario en Internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida''. Esto se consigue dotando a la Web de más significado y, por lo tanto, de más semántica, lo que permite obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla. Esta Web extendida y basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante.

Existen también otras definiciones como la que establece (Castells, 2003), en la que se define la Web Semántica como ''un área pujante en la confluencia de la Inteligencia Artificial y las tecnologías web que propone introducir descripciones explícitas sobre el significado de los recursos, para permitir que las propias máquinas tengan un nivel de comprensión de la Web suficiente como para hacerse cargo de una parte, la más costosa, rutinaria, o físicamente inabarcable, del trabajo que actualmente realizan manualmente los usuarios que navegan e interactúan con la Web''. Como se puede comprobar, esta definición posee muchos puntos en común con la proporcionada en un primer momento, con la salvedad de que en ésta se

establece una relación con la Inteligencia Artificial que ha sido objeto de diferentes debates, ya que algunos expertos lo consideran dentro de este área, otros como una relación que se ha establecido a medio camino y, por último, otro grupo que lo considera totalmente fuera de esos límites. Este debate posee diferentes puntos de vista, sin embargo no se encuentra dentro de los objetivos de esta investigación el solucionar esta disyuntiva.

Para comprender mejor esta representación de la realidad se estudiará un caso práctico centrado en el ámbito del turismo. Un usuario accede a una página de un buscador para localizar empresas de alquiler de coches cercanas a un cierto hotel de destino. Actualmente el ordenador no entiende el significado semántico de la búsqueda "Alquilar un Coche cerca del Hotel", sino que en todo caso realizará una búsqueda a nivel léxico de las palabras.

Debido a esto, si el usuario desea localizar la agencia de alquiler de vehículos, en primer lugar, deberá localizar la ciudad donde se alojará; posteriormente, deberá acceder a las páginas que contengan empresas de alquiler de vehículos de la propia ciudad; y, por último, una vez localizada la agencia de alquiler deseada, consultará las tarifas de los vehículos y su disponibilidad temporal. Por tanto, con la Web actual no hay modo posible de que este proceso se pueda automatizar. Es aquí donde entra el potencial de la Web Semántica, proporcionando solución a estos problemas a partir una serie de tecnologías, conceptos y paradigmas que la han llevado a ser bautizada como el “Internet del futuro”.

Efectivamente, las tecnologías semánticas han sido señaladas en múltiples ocasiones como el futuro de la Web (Benjamins et al., 2008) y como una nueva forma de apoyo y soporte al conocimiento (Vossen, Lytras, & Koudas, 2007), (D. Fensel & Musen, 2001) en un amplio rango de dominios entre los que se incluyen los sistemas de recomendación (García-Crespo, Colomo-Palacios, Gómez-Berbís, & Ruiz-Mezcua, 2010), biomedicina (Garcia-Sanchez, Fernandez-Breis, Valencia-Garcia, Gomez, & Martinez-Bejar, 2008), seguridad (García-Crespo, Gómez-Berbís, Colomo-Palacios, & Alor-Hernández, 2011), innovación (Colomo-Palacios, Ricardo, García-Crespo, Soto-Acosta, Ruano-Mayoral, & Jiménez-López, 2010) o la Ingeniería del Software (García-Crespo, Colomo-Palacios, Gómez-Berbís, & Mencke, 2009), por citar algunos de los trabajos más relevantes en este ámbito.

La principal tecnología de representación del conocimiento usada en la Web Semántica es la ontología (Dieter Fensel, 2003), que formaliza dicho significado y facilita la búsqueda de contenidos e información (Jiang & Tan, 2009) así como mejorar el rastreo en la Web (Zheng, Kang, & Kim, 2008) gracias a que proporciona una marca común que permite la integración, compartición y reutilización de los datos desde múltiples fuentes. Se pueden encontrar un gran número de definiciones de ontología. Una de las más relevantes es la de (Studer, Benjamins, & Fensel, 1998) que afirma que ''una ontología es una especificación formal y explícita de una conceptualización compartida''. En este contexto, formal se refiere a la necesidad de ontologías comprensibles por la máquina. Esta definición enfatiza la necesidad de alcanzar un acuerdo para establecer la conceptualización compartida. Las ontologías proporcionan un vocabulario común de un área y define, con diferentes niveles de formalidad, el significado de términos y las relaciones que se establecen entre ellos. El conocimiento en las ontologías se encuentra principalmente formalizado empleando cinco tipos de componentes: clases, relaciones, funciones, axiomas e instancias (Gruber, 1993).

Existen numerosas ontologías, cada una con diferente propósito que cubren dominios concretos. Este es el caso de BORO (Business Object Reference Ontology) (Partridge & Stefanova, 2001) una ontología que intenta ser adecuada como base para la facilitación, entre otras cosas, en interoperabilidad semántica de los sistemas operacionales de la empresa. Otro ejemplo de un dominio completamente diferente es el presentado en el ámbito de la seguridad en (Vorobiev & Bekmamedova, 2010), donde una aproximación mediante la utilización de ontologías, permite reforzar la seguridad de la información. Más cercano a los objetivos de la tesis se encuentra (Youseff, Butrico, & Da Silva, 2008) centrado en el dominio del Cloud Computing, que es uno de los primeros intentos para establecer una ontología de la nube, intentando proporcionar un mejor entendimiento de la tecnología para ser capaz de desarrollar portales más eficientes y puertas para los entornos de la nube.

Relacionado con la aplicación de tecnologías semánticas, es importante comentar que los ámbitos en los que se usan son variados y cubren un amplio espectro desde aplicaciones recientes de técnicas de Web Semántica para el control de acceso a redes (Fitzgerald, Foley, & Foghlu, 2009), hasta la integración y modelos de análisis para sistemas de información (A. Sheth & Ramakrishnan, 2003) o incluso aplicaciones de e-business como en (Singh, Iyer, & Salam, 2005). La Web Semántica también ha sido utilizada en la creación de modelos para interoperabilidad en la Web como presenta (Melnik, Rahm, & Sosna, 2001) donde se usa un enfoque basado en capas.

Centrándose en entornos de Cloud Computing, es posible encontrar algunos trabajos que emplean tecnologías semánticas para aprovechar su potencial y características, pero en (Nyre & Jaatun, 2009) se presenta una interesante encuesta sobre el potencial de las nubes semánticas si fuese posible solucionar los retos que se plantean en cuanto a privacidad. Otras aproximaciones se han tomado en (Garcia-Sanchez et al., 2009) y (Garcia-Sanchez et al., 2010), donde se consiguen objetivos más cercanos a los planteados en este trabajo y en los que las tecnologías semánticas toman un rol principal en la consecución de las metas, combinando sus capacidades con las ventajas del SaaS y el Cloud Computing y sentando las bases para la construcción de plataformas cloud capaces de soportar todo este potencial y ponerlo a disposición del usuario.

2.4.1 **Ontologías**

Heredadas de la investigación en Inteligencia Artificial, las ontologías pretenden describir el conocimiento (conceptos, atributos y relaciones) de un dominio dado. Fueron diseñadas para poderse compartir, de forma que distintas aplicaciones manejen los mismos conocimientos y puedan colaborar entre sí.

Actualmente existen muchas definiciones de una ontología. Una de las más conocidas es la de (Gruber, 1993), que define una ontología como ''una especificación explícita de una conceptualización''. Al desplazarse hacia una definición más práctica, se puede decir que una ontología es una descripción formal de conceptos básicos (clases) en un dominio, propiedades (atributos) de cada uno de estos conceptos y restricciones en dichas propiedades, así como las relaciones entre las clases (Noy & Mcguinness, 2001). Además de ello constituye una

tecnología clave para el desarrollo de la Web Semántica, ya que une la comprensión simbólica humana y la representación del conocimiento con el procesamiento por ordenador (Davies, Studer, & Warren, 2006).

Las ontologías dan cuenta del conocimiento del dominio en términos estáticos. Dicha tecnología se ha utilizado ya para representar conocimiento en distintos tipos de dominios, como los clínicos (Schulz, Romacker, Faggioli, Freiburg, & Hahn, 1999; Shahar & Musen, 1996), las memorias organizacionales (Dieng, Corby, Giboin, Ribière, & Lucioles, 1998), (Schwartz, 1999), la gestión del conocimiento (J.T. Fernández-Breis, 2003; Sure, Prof, & Studer, 2003), bioinformática (Stevens, Goble, & Bechhofer, 2000) e incluso e-learning (Jesualdo Tomás Fernández-Breis, Castellanos-Nieves, & Valencia-García, 2009). Las ontologías permiten que el conocimiento que éstas contienen pueda reutilizarse y compartirse, por lo que su uso conlleva una reducción del esfuerzo necesario para implementar sistemas expertos.

Desde los años 90 ha sido un tema puntero de investigación, y han sido estudiadas en varias comunidades científicas como ingeniería del conocimiento, procesamiento de lenguaje natural o representación de conocimiento. El motivo de su creciente popularidad es principalmente lo que prometen: una comprensión compartida y común de un dominio que puede ser comunicado entre personas y aplicaciones informáticas. El uso de las ontologías, en definitiva, ofrece una oportunidad de mejorar las posibilidades de realizar cualquier tarea relacionada con la gestión de información y conocimiento.

Existen diferentes lenguajes para la creación de ontologías, aunque la W3C (World Wide Web Consortium) recomienda el uso de OWL (Web Ontology Language) (McGuinness & van Harmelen, 2004). En los siguientes apartados se ampliará la información a este respecto y se analizarán en detalle las propiedades más relevantes para el curso de la investigación.

2.4.2 Razonamiento e inferencia lógica

La lógica es la disciplina que estudia los principios del razonamiento. Un razonamiento es una actividad mental que consiste en pasar de unas proposiciones a otras, partiendo de lo ya conocido o de lo que creemos conocer (premisas) a lo desconocido o menos conocido (conclusión). Un razonamiento también es el resultado de dicha actividad, es decir, un conjunto de proposiciones enlazadas entre sí que dan apoyo o justifican una idea. El razonamiento se corresponde con la actividad verbal de argumentar. En otras palabras, un argumento es la expresión verbal de un razonamiento.

El razonamiento nos permite ampliar nuestros conocimientos sin tener que apelar a la experiencia. Un razonador automático permite deducir o inferir conclusiones a partir de un conocimiento determinado, haciendo explícito un conocimiento implícito.

Existen en la actualidad diversas formas y herramientas para la gestión de este tipo de operaciones. Entre las más significativas se encuentra KAON2, el cual provee un motor de inferencia para la resolución de consultas conjuntas sobre OWL-DL, SWRL ("Semantic Web Rule Language") y F-Logic (derivado de "Frame Logic"). Expresadas utilizando la sintaxis SPARQL. Su predecesor, KAON, ofrecía similares características pero sobre ontologías definidas únicamente en RDF.

Así, dentro de la Web Semántica, la lógica se puede utilizar para descubrir conocimiento que está dado implícitamente. Los lenguajes RDF y OWL se pueden ver como una especialización de esta lógica de predicados. Más información sobre estos y más lenguajes será proporcionada a lo largo de este documento.

2.4.3 Lenguajes empleados en Web Semántica

2.4.3.1 RDF

El Resource Description Framework (RDF) es un estándar (aunque siendo estrictos en la definición se debe entender como una recomendación de la W3C) para la descripción de recursos (Klyne & Carroll, 2004). En torno a los recursos existe aún un debate acerca de qué es considerado un recurso y qué no. En el ámbito de esta investigación se considera un recurso cualquier entidad que se pueda identificar, desde una persona o una página web hasta un número.

Como ya se ha comentado anteriormente, en un principio la World Wide Web se construyó para el uso humano, y a pesar de que todo en ella era legible por máquinas, estos datos todavía no son legibles por ellas. Es muy difícil automatizar tareas en la Web debido en gran medida al volumen de información que contiene, lo que hace imposible gestionarla manualmente. La solución que se propone es el uso de metadatos que permitan describir los datos contenidos en la Web. Los metadatos son ''datos sobre los datos'' (por ejemplo, un catálogo de biblioteca) o concretamente en el contexto de esta especificación ''datos que describen recursos Web''. La distinción entre datos y metadatos no es incuestionable; es una diferencia creada en primera instancia por una aplicación particular, y muchas veces el mismo recurso se interpretará de ambas formas simultáneamente.

El RDF es una base para procesar metadatos; proporciona interoperabilidad entre aplicaciones que intercambian información legible por máquina en la Web. RDF destaca por la facilidad para habilitar el procesamiento automatizado de los recursos Web. Puede utilizarse en distintas áreas de aplicación; recuperación de recursos para proporcionar mejores prestaciones a los motores de búsqueda, catalogación para describir el contenido y las relaciones de contenido disponibles en un sitio Web, una página web, o una biblioteca digital particular, por los agentes de software inteligentes para facilitar el intercambio y para compartir conocimiento, etc. RDF junto con las firmas digitales será la clave para construir una ''Web de confianza'' para el comercio electrónico, la cooperación y otras aplicaciones.

En esta sección se presenta de forma resumida un modelo para representar metadatos en RDF así como una sintaxis para codificar y transmitir estos metadatos de tal forma que maximicen la interoperabilidad de servidores y clientes web desarrollados independientemente. La sintaxis presentada aquí utiliza el Extensible Markup Language (XML). Uno de los objetivos de RDF es hacer posible especificar la semántica para las bases de datos en XML de una forma normalizada e interoperable. RDF y XML son complementarios: RDF es un modelo de metadatos y sólo dirige por referencia muchos de los aspectos de codificación que requiere el almacenamiento y transferencia de archivos. Para estos aspectos, RDF cuenta con el soporte

de XML. Es importante también entender que esta sintaxis XML es sólo una sintaxis posible para RDF y que pueden surgir formas alternativas para representar el mismo modelo de datos RDF.

El objetivo general de RDF es definir un mecanismo para describir recursos que no cree ninguna asunción sobre un dominio de aplicación particular, ni defina (a priori) la semántica de algún dominio de aplicación. La definición del mecanismo debe ser neutral con respecto al dominio, sin embargo el mecanismo debe ser adecuado para describir información sobre cualquier dominio.

Para facilitar la definición de metadatos, RDF cuenta con un sistema de clasificación muy parecido a los sistemas de programación y modelado orientado a objetos. Una colección de categorías (producida normalmente para un propósito o dominio específico) denominada schema. Las categorías se organizan en una jerarquía, y proporcionan extensibilidad a través de un refinamiento de subcategorías. Así, para crear un esquema ligeramente diferente de uno existente no es necesario ''reinventar'', pero se pueden facilitar modificaciones incrementales al esquema base. A través de la compartición del esquema RDF se soporta la reutilización de definiciones de metadatos. Gracias a la extensibilidad incremental de RDF, los agentes que procesen metadatos serán capaces de trazar los orígenes del esquema que no conocen con respaldo para conocer el esquema y realizar acciones significativas en los metadatos que no han sido diseñados inicialmente para procesar. La capacidad de compartir y la extensibilidad de RDF permiten a los creadores de metadatos usar múltiples cambios de la categoría de objetos para unir definiciones, para proporcionar múltiples presentaciones a sus datos, haciendo uso del trabajo de otros. En síntesis, es posible crear objetos específicos de datos basados en diversos esquemas de distintas fuentes. Los esquemas pueden escribirse en RDF; un documento que acompaña a esta especificación [RDFSchema], describe un conjunto de propiedades y clases para describir esquemas RDF.

Como resultado de la reunión de distintas comunidades que están de acuerdo en los principios básicos de la representación y transposición de metadatos, RDF está influido por varias fuentes diferentes. Las principales influencias provienen de la propia Comunidad de Normalización de la Web en forma de metadatos HTML y PICS, la comunidad bibliotecaria, la comunidad de los documentos estructurados en forma de SGML y sobre todo XML, y también de la comunidad de representación del conocimiento (KR). También han contribuido al diseño de RDF otras áreas de la tecnología; incluidos los lenguajes de modelado y programación orientada a objetos, así como las bases de datos. Mientras RDF surge de la comunidad KR, la bibliografía relacionada con este campo advierte que RDF no especifica un mecanismo para el razonamiento. RDF puede definirse como un sistema simple. Un mecanismo de razonamiento debe construirse sobre este sistema de referencia.

2.4.3.1.1 RDF Básico

Modelo RDF básico

El fundamento o base de RDF es un modelo para representar propiedades designadas y valores de propiedades. El modelo RDF se basa en principios perfectamente establecidos de varias comunidades de representación de datos. Las propiedades RDF pueden recordar a atributos de recursos y en este sentido corresponden con los tradicionales pares de atributo-valor. Las propiedades RDF representan también la relación entre recursos y por lo tanto, un modelo RDF puede parecer un diagrama entidad-relación. En la terminología del diseño orientado a objetos, los recursos corresponden con objetos y las propiedades corresponden con objetos específicos y variables de una categoría.

El modelo de datos de RDF es una forma de sintaxis-neutral para representar expresiones RDF. La representación del modelo de datos se usa para evaluar la equivalencia en significado. Dos expresiones RDF son equivalentes si y sólo si sus representaciones del modelo de datos son las mismas. Esta definición de equivalencia permite algunas variaciones sintácticas en expresiones sin alterar el significado.

El modelo de datos básico consiste en tres tipos de objetos:

1. **Recursos**. Todo aquello descrito por expresiones RDF se denominan recursos. Los recursos se designan siempre por URIs más identificadores de anclas opcionales. Cualquier cosa puede tener un URI; la extensibilidad de URIs permite la introducción de identificadores para cualquier entidad imaginable.
2. **Propiedades**. Una propiedad es un aspecto específico, característica, atributo, o relación utilizado para describir un recurso. Cada propiedad tiene un significado específico, define sus valores permitidos, los tipos de recursos que puede describir y sus relaciones con otras propiedades.
3. **Sentencias** (declaraciones, enunciados). Un recurso específico junto con una propiedad denominada, más el valor de dicha propiedad para ese recurso es una sentencia RDF [RDF statement]. Estas tres partes individuales de una sentencia se denominan, respectivamente, sujeto, predicado y objeto.

    - **Sujeto**. Es el recurso desde el que parte el arco en el grafo formado.
    - **Predicado**. Es la propiedad que etiqueta el arco en el grafo.
    - **Objeto**. Es el recurso o literal al que llega el arco en el grafo. Puede ser otro recurso o pude ser un literal; es decir, un recurso (especificado por un URI), una cadena simple de caracteres u otros tipos de datos primitivos definidos por XML. En términos RDF, un literal puede comprender en su contenido marcado XML pero ya no puede valorarse más por un procesador RDF.

Sintaxis RDF básica

El modelo de datos RDF proporciona un marco abstracto y conceptual para definir y utilizar metadatos y por ello necesita también una sintaxis concreta para crear e intercambiar metadatos. Esta especificación de RDF utiliza XML codificado como su sintaxis de intercambio. RDF necesita también la XML namespace facility para asociar con precisión cada propiedad con el esquema que define dicha propiedad.

Las descripciones sintácticas usan la forma de notación Extended Backus-Naur Form (EBNF) ampliamente conocida y extendida. Notación, de XML para describir los elementos sintácticos esenciales en RDF. EBNF aquí se centra en la legibilidad humana, en particular la cursiva ''rdf'' se usa para representar un prefijo de namespace variable más que la notación BNF más precisa '' $<$ NSprefix $:...$''. La condición es que la propiedad y el tipo de nombres en las etiquetas finales coincidan con los nombres en las etiquetas iniciales correspondientes implicadas en las reglas de XML. Todas la flexibilidades sintácticas de XML están incluidas implícitamente, p. ej. las reglas de espacio en blanco, citar usando tanto comillas simple (') como comillas dobles (''), character escaping, case sensibility, y marcado del lenguaje.

Esta especificación define dos sintaxis XML para codificar una instancia del modelo de datos:

1. La sintaxis serializada expresa las capacidades totales del modelo de datos de una forma muy regular.
2. La sintaxis abreviada incluye términos adicionales que proporcionan una forma más compacta para representar un subconjunto del modelo de datos.

Los intérpretes de RDF se han anticipado a implementar ambas sintaxis, así los autores o creadores de metadatos pueden mezclar ambas libremente.

2.4.3.2 OWL

La Web Semántica es una visión del futuro de la Web donde la información está proporcionando un significado explícito, permitiendo que las máquinas puedan procesar automáticamente e integrar la información disponible en la Web. La Web Semántica se basará en la capacidad de XML para definir esquemas de etiquetas a medida y en la aproximación flexible de RDF para representar datos. El primer nivel requerido por encima de RDF para la Web Semántica es un lenguaje de ontologías que pueda describir formalmente el significado de la terminología usada en los documentos Web. Si se espera que las máquinas hagan tareas útiles de razonamiento sobre estos documentos, el lenguaje debe ir más allá de las semánticas básicas del RDF Schema.

Con este fin nace OWL. Sus siglas pertenecen al acrónimo en inglés de Web Ontology Language y se trata de un lenguaje de marcado ideado para publicar y compartir datos mediante ontologías que forma parte de un conjunto creciente de recomendaciones del W3C relacionadas con la Web Semántica (McGuinness & van Harmelen, 2004).

OWL está pensado para ser usado cuando la información contenida en los documentos necesita ser procesada por las aplicaciones, al contrario que en las situaciones donde el

contenido sólo necesita ser presentado a los humanos. OWL puede ser usado para representar explícitamente el significado de términos en vocabularios y las relaciones entre esos términos mediante ontologías. OWL tiene mayor capacidad para expresar significado y semántica que XML, RDF, y RDF-S, y, de este modo, OWL va más allá de estos lenguajes en su capacidad para representar contenido interpretable por un ordenador en la Web. OWL es una revisión del Lenguaje de Ontologías Web DAML+OIL incorporando lecciones aprendidas a partir del diseño y aplicación de DAML+OIL.

OWL añade vocabulario para describir propiedades y clases: entre otros, relaciones entre clases, cardinalidad, igualdad, más tipos de propiedades, características de propiedades, clases enumeradas, etc.

OWL tiene tres sublenguajes, con un nivel de expresividad creciente: OWL Lite, OWL DL, y OWL Full. A continuación se proporciona una breve descripción de cada uno de ellos:

- **OWL Lite** está diseñado para aquellos usuarios que necesitan principalmente una clasificación jerárquica y restricciones simples. Por ello, a la vez que admite restricciones de cardinalidad, sólo permite establecer valores cardinales de 0 ó 1. OWL Lite proporciona además una ruta rápida de migración para tesauros y otras taxonomías. Tiene también una menor complejidad formal que OWL DL.
- **OWL DL** está diseñado para aquellos usuarios que quieren la máxima expresividad conservando completitud computacional (se garantiza que todas las conclusiones sean computables), y resolubilidad (todos los cálculos se resuelven en tiempo finito). OWL DL incluye todas las construcciones del lenguaje de OWL, pero sólo pueden ser usados bajo ciertas restricciones (por ejemplo, mientras una clase puede ser una subclase de otras muchas clases, una clase no puede ser una instancia de otra). OWL DL es denominado de esta forma debido a su correspondencia con la lógica de descripción (Description Logics, en inglés), un campo de investigación que estudia la lógica que compone la base formal de OWL.
- **OWL Full** está dirigido a usuarios que quieren máxima expresividad y libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser considerada simultáneamente como una colección de clases individuales y como una clase individual propiamente dicha. OWL Full permite una ontología para aumentar el significado del vocabulario preestablecido (OWL o RDF). Es poco probable que cualquier software de razonamiento sea capaz de obtener un razonamiento completo para cada característica de OWL Full.

Cada uno de estos sublenguajes es una extensión de su predecesor más simple, respecto a lo que puede ser expresado legalmente y a la validación de sus conclusiones. El siguiente grupo de relaciones se mantienen, pero las relaciones inversas no se mantienen.

- Cada ontología legal de OWL Lite es una ontología legal de OWL DL
- Cada ontología legal de OWL DL es una ontología legal de OWL Full
- Cada conclusión válida de OWL Lite es una conclusión válida de OWL DL
- Cada conclusión válida de OWL DL es una conclusión válida de OWL Full

Los desarrolladores de ontologías que adoptan OWL deberían considerar cuál es el sublenguaje que mejor se adapta a sus necesidades. La elección entre OWL Lite y OWL DL depende de las necesidades de los usuarios sobre la expresividad de las construcciones, proporcionando OWL DL las más expresivas. La elección entre OWL DL y OWL Full depende principalmente de las necesidades de los usuarios sobre los recursos de metamodelado del esquema RDF (por ejemplo, definir clases de clases, o definir propiedades de clases). Cuando se usa OWL Full en comparación con OWL DL, el soporte en el razonamiento es menos predecible, ya que no existen en este momento implementaciones completas de OWL Full.

OWL Full puede ser considerada como una extensión de RDF, mientras que OWL Lite y OWL DL pueden ser consideradas como extensiones de una visión restringida de RDF. Cada documento OWL (Lite, DL, Full) es un documento RDF, y cada documento RDF es un documento de OWL Full, pero sólo algunos documentos RDF serán legalmente documentos OWL Lite u OWL DL. Por este motivo, se ha de tener cuidado cuando se quiera migrar un documento de RDF a OWL. Cuando se considere que la expresividad de OWL DL u OWL Lite es adecuada, han de tomarse algunas precauciones para asegurar que el documento RDF original cumple con las restricciones adicionales impuestas por OWL DL y OWL Lite. Entre otras, cuando una URI es utilizada como nombre de una clase, debe indicarse explícitamente que esta URI debe ser una clase del tipo owl:Class (al igual que para las propiedades), cada individuo debe estar definido como perteneciente al menos a una clase (incluso sólo con objetos owl:Thing) y las URI usadas para las clases, propiedades e individuos deben ser disjuntos entre ellos.

2.4.3.3 SWRL

SWRL (Semantic Web Rule Language) es una propuesta de lenguaje basado en reglas para Web Semántica en la que se combinan sublenguajes de OWL DL y OWL Lite con Rule Markup Language. Extiende OWL incluyendo cláusulas de Horn que pueden ser combinadas con la base de conocimiento de OWL. Ofrece una sintaxis abstracta de alto nivel que amplía la proporcionada por OWL (Horrocks et al., 2004).

Sus especificaciones fueron desarrolladas en mayo de 2004 para la W3C por parte del National Research Council de Canada, Network Inference y de la Universidad de Stanford conjuntamente con el consorcio US/EU y el Agent Markup Language Committee.

En cuanto a las reglas, una ontología en sintaxis abstracta de OWL contiene una secuencia de axiomas y hechos. Las reglas se forman con un antecedente y un consecuente, el cual podría estar formado por un conjunto de átomos o encontrarse vacío. Un axioma puede también utilizar referencias URI que sirven para identificar una regla.

rule ::= 'Implies(' [URIreference] { annotation } antecedent consequent ')'

antecedent ::= 'Antecedent(' { atom } ')'

consequent ::= 'Consequent(' { atom } ')'

Antecedente y consecuente pueden estar formados por uno o más átomos. Si el antecedente se encuentra vacío implica verdad absoluta, si es el consecuente el que se encuentra vacío

implica la falsedad del antecedente. Los átomos en estas reglas pueden ser de la forma: C(x), P(x,y),

sameAs(x,y) o differentFrom(x,y), donde C es una descripción OWL, P es una propiedad OWL, y ''x'',''y'' son también variables individuales de OWL o valores de datos OWL.

Como se ha comentado con anterioridad, la sintaxis empleado por SWRL es bastante abstracta, hecho que facilita el acceso y evaluación del lenguaje. Por estos motivos posee muchas similitudes con la sintaxis EBNF:

- Las alternativas son separadas por una barra vertical ($|$) o escritas en diferentes reglas.
- Los componentes que pueden ocurrir al menos una vez están entre corchetes ([...])
- Los componentes que pueden ocurrir n veces entre llaves ({...})
- Los espacios en blanco son ignorados.

Los átomos pueden referirse a individuos, literales, variables individuales o variables de datos. Las variables se tratan como universalmente cuantificadas, con su alcance limitado hacia una regla dada. Como suele ocurrir, sólo las variables que ocurren en el antecedente de una regla pueden ocurrir en el consecuente. Esta condición no restringe, sin embargo, la expresividad del lenguaje.

La sintaxis abstracta EBNF es consistente con la especificación OWL, sin embargo no es particularmente fácil de leer. Para solucionar este problema, se trata de normalizar la forma de expresar las reglas.

- Tanto el Antecedente como el Consecuente son conjunciones de átomos (a1 and...and an).
- Las variables son marcadas con un signo de ''?'' como prefijo.
- Ejemplo: parent(?x,?y) and brother(?y,?z) => uncle(?x,?z)
- Las relaciones funcionales pueden escribirse de dos formas:
  1. op:numeric-add(?x,3,?z)
  2. ?x = op:numeric-add(3,?z)

Seguidamente se muestran algunos ejemplos que ayudarán a comprender mejor su funcionamiento:

1. Un uso simple de las reglas es la afirmación de que las propiedades hasParent y hasBrother implican hasUncle:

   - hasParent(?x1,?x2) and hasBrother(?x2,?x3) => hasUncle(?x1,?x3)

   Con una sintaxis abstracta se escribiría:

   - Implies(Antecedent(hasParent(I-variable(x1)Ivariable(x2))hasBrother(I-variable(x2) Ivariable(x3))) Consequent(hasUncle(Ivariable(x1) I-variable(x3))))

2. Otro ejemplo sería la afirmación de que los estudiantes son personas:
   - Student(?x1) => Person(?x1)

Que podría escribirse como:

- Implies(Antecedent(Student(I-variable(x1)))Consequent(Person(I-variable(x1))))

Aunque al ser muy simple se podría escribir directamente en OWL:

- Class(Student partial Person)
- SubClassOf(Student Person)

Existen diferentes implementaciones válidas para utilizar SWRL. Las más relevantes son las siguientes:

1. SWRLTab. Es una extensión de Protégé que soporta la edición y ejecución de reglas SWRL
2. R2ML (REWERSE Rule Markup Language)
3. Bossam. Motor de reglas que soporta SWRL
4. Hoolet. Implementación de un razonador OWL-DL que utiliza un probador de primer orden que soporta SWRL
5. Pellet. Razonador de código abierto hecho en Java OWL DL con soporte para SWRL
6. KAON2. Es una infraestructura para la gestión de ontologías en OWL-DL, SWRL y F-Logic
7. RacerPro. Soporta el procesado de reglas en una sintaxis basada en SWRL para traducirlo en reglas nRQL

Se debe de hacer especial mención al hecho de que los razonadores no soportan la especificación completa debido a que el razonamiento puede llegar a ser indecidible. Existen tres tipos de aproximaciones, como se acaba de mencionar. La primera traduce SWRL a lógica de primer orden (Hoolet) y realiza tareas de razonamiento y demostración con un experimentador de teoremas. La segunda opción se encarga de traducir OWL-DL a reglas y pasar esas reglas a un motor forward chaining (Bossam). Esta aproximación no permite cubrir la total expresividad de OWL-DL debido a numerosas incompatibilidades entre los formalismos de Description Logic y Cláusulas de Horn. Por último, se tiene la expansión del razonador existente OWL-DL basado en el algoritmo de tableaux (Pellet).

Para finalizar esta sección, comentar que los DLPs (Description Logic Programs) forman otra propuesta para la integración de reglas y OWL. Comparado con DLPs, SWRL escoge un camino de integración diametralmente opuesto. DLP se forma de la intersección de la lógica de Horn y OWL mientras que SWRL es la unión de ambos, formando dos opciones con posibilidades diferentes.

2.4.4 **Herramientas de trabajo para Web Semántica**

2.4.4.1 JENA

Jena es un framework open-source para la construcción de aplicaciones Java relacionadas con la Web Semántica. Jena1 fue lanzado en el año 2000 y se produjeron alrededor de 10.000 descargas desde la red. Jena2, con una arquitectura interna revisada y características nuevas, apareció en Agosto de 2003 y en este caso se alcanzaron las 7.000 descargas.

La principal contribución realizada por Jena1 (McBride, 2001) fue la API para manipular grafos RDF. Alrededor de esta API, Jena1 proporcionaba varias herramientas incluyendo módulos de entrada y salida para RDF/XML, N3 y N-triple, además del lenguaje de consultas RDQL. Mediante el uso de esta API el usuario puede elegir almacenar los grafos RDF en memoria o en almacenes persistentes. Jena1 proporciona una API adicional para la manipulación de DAML+OIL.

Del feedback proporcionado por los usuarios de Jena1, se extraía la conclusión de que se debía mejorar la integración el soporte DAML+OIL y el soporte RDF para permitir, por ejemplo, el almacenamiento de modelos DAML dentro de las bases de datos. También se comprobó que era demasiado difícil añadir mayores implementaciones del API de Jena1.

En respuesta a las dificultades mencionadas, Jena2 posee una arquitectura más desarrollada que Jena1. Los dos objetivos principales de la arquitectura de Jena 2 son:

1. Múltiple y flexible presentación de grafos RDF para el programador de aplicaciones. Esto permite que los datos del grafo sean accesibles y manipulables a través de interfaces de alto nivel.
2. Una vista minimalista y simple del grafo RDF al programador del sistema para manipular los datos como triples. Esto se vuelve particularmente útil para razonamiento con RDFS y OWL.

Jena soporta un Semantic Web Query Language, RDQL (Miller, Seaborne, & Reggiori, 2002), que puede ser utilizado en grafos materializados o en los resultados virtuales de razonamiento con RDFS u OWL. Las consultas completas pueden ser pasadas dentro de capas inferiores del grafo de manera que los grafos de la base de datos puedan aprovechar las ventajas de la optimización de SQL. Una tercera presentación de la interfaz, RDF WebAPI, proporcionar clientes web para acceder a grafos RDF basados en consultas. Este tipo de acceso también se encuentra disponible tanto en la interfaz del sistema como en la de aplicación y actúa como unificador temático de la arquitectura.

2.4.4.1.1 Formas de trabajar con ontologías en Jena

El almacenamiento en memoria es la forma clásica con la que suelen trabajar habitualmente las aplicaciones. En el caso de aplicaciones que involucran el manejo de ontologías, equivale a tener el modelo OWL (ontología) almacenado en memoria principal, como si se tratase de una

variable de programa. Una forma de trabajar con ontologías en Jena es declarando una variable OntModel e ir construyendo el modelo en la propia aplicación (creación de clases, subclases, propiedades, restricciones, etc.). Con Jena también es posible cargar un modelo a partir de una fuente local, por ejemplo, un fichero que contenga una ontología definida en el lenguaje de marcado OWL.

El almacenamiento persistente surge ante la necesidad de guardar datos de programa de forma duradera para recuperarlos en otro momento. Obviamente, el almacenamiento persistente en bases de datos supone grandes ventajas sobre el almacenamiento en memoria y el almacenamiento tradicional en el sistema de ficheros. Para ejecutar operaciones sobre bases de datos en una aplicación Java, es necesario JDBC (Java Database Connectivity). La API JDBC es una colección de interfaces Java y métodos de gestión de manejadores de conexión para cada modelo específico de base de datos. En este ámbito, Jena es capaz de trabajar con ontologías almacenadas de forma persistente en una base de datos utilizando el driver JDBC.

2.5 Servicios Web

Se puede definir un servicio web como una aplicación autocontenida, auto-descriptiva y modular, basada en contenidos XML que pueden publicarse, localizarse e invocarse en la Web. Estos servicios se localizan mediante registros y pueden ampliarse con metadatos descriptivos para los consumidores del servicio (Alonso, Casati, Kuno, & Machiraju, 2004). Con la aparición de los servicios web se abre la posibilidad de conseguir la automatización de tareas entre plataformas y servicios diferentes.

Los servicios web añaden una capa de abstracción sobre la web tradicional que permite extenderla con un elemento de dinamismo, convirtiéndola en un conjunto de aplicaciones accesibles desde cualquier sitio de la red. Esta nueva arquitectura ha incidido en la forma de desarrollar aplicaciones permitiendo hacerlas más escalables e interoperables.

La Tabla 3 muestra algunas de las novedades que aporta este modelo comparado con el basado en componentes que se ha venido utilizando:

Modelos basados en componentes	Modelos de servicios web
Aplicaciones **fuertemente** acopladas (alta dependencia entre sistemas)	Aplicaciones **débilmente** acopladas (baja dependencia entre sistemas)
Diseñados para procesos **internos** de la empresa	Diseñado para procesos **externos** a la empresa
Diferentes protocolos y tecnologías (DCOM, CORBA...)	Protocolos y tecnologías **comunes** (XML, SOAP, WSDL...)

Tabla 3. Comparativa entre modelos basados en componentes y servicios web

El escenario común de funcionamiento de los servicios web consiste principalmente en la interacción de tres componentes software. A continuación la Figura 14 presenta cada uno de los elementos que intervienen:

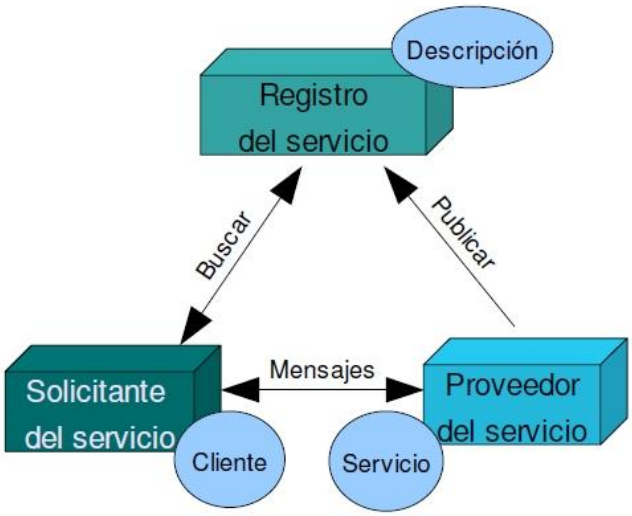


Figura 14. Componentes de un servicio web (mundointernet.es)

1. **Proveedor del servicio**. Es un servidor que hospeda al propio servicio y que provee su interfaz, comúnmente en el nivel de aplicación sobre un protocolo de transporte. Este componente interviene en la publicación del servicio y en el intercambio de mensajes con el solicitante del servicio.
2. **Registro del servicio**. Es la aplicación que mantiene la información necesaria para identificar el servicio e interviene en operaciones de búsqueda y descubrimiento de servicios.
3. **Solicitante del servicio**. Es la aplicación que requiere de la funcionalidad que ofrece un servicio concreto y se conecta con el proveedor para operar con el servicio.

Para asegurar la interoperabilidad de los servicios, es necesario que este escenario se construya sobre estándares. Dichos estándares son definidos por las organizaciones OASIS (Organization for the Advancement of Structured Information Standards) y W3C (World Wide Web Consortium). A continuación, se describe la pila de protocolos y lenguajes comúnmente aceptados y las distintas capas de abstracción de las que se compone un sistema típico de servicios web:

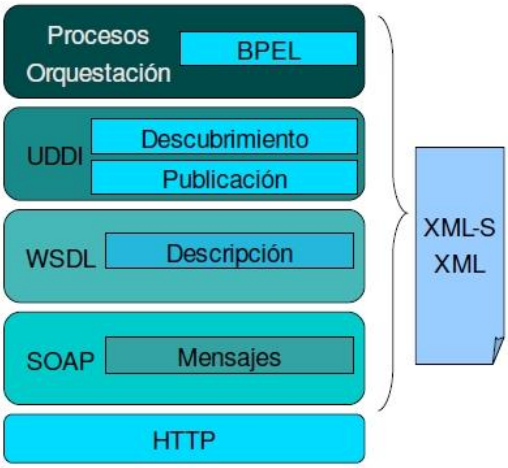


Figura 15. Protocolos y tecnologías de los servicios web (mundointernet.es)

La capa más baja de la pila se identifica con la capa de transferencia de mensajes. En ella se definen las direcciones únicas (URI - Uniform Resource Identifier) de los recursos. Además del protocolo HTTP (HyperText Transfer Protocol), otros protocolos son utilizados a este nivel como SMTP (Simple Mail Transfer Protocol), JMS (Java Message Service), etc.

La segunda capa consiste en el protocolo de intercambio de mensajes que bien pueden estar en SOAP (Simple Object Access Protocol) basado en XML (Extensible Markup Language) o en arquitectura REST (Representational State Transfer) basado en mensajes y operaciones HTTP.

En el siguiente nivel se encuentra la descripción del servicio web donde se pueden diferenciar dos subniveles principales:

- Nivel alto. Se definen las operaciones que ofrece el servicio, en función de sus mensajes de entrada y salida.
- Nivel menos abstracto. Se define cómo interaccionan WSDL (Web Service Description Language) y SOAP, o generalizando, la capa de descripción del servicio con la capa de paso de mensajes.

El siguiente nivel de abstracción entra en escena cuando se parte de un servicio web previamente definido y es necesario publicarlo. El protocolo utilizado para ello es UDDI (Universal Description, Discovery and Integration). UDDI organiza de un modo jerárquico las compañías, sus servicios y la información para acceder a dichos servicios. Algunos servidores UDDI son jUDDI (Apache Software Foundation - 2008) o Novell UDDI Server (Novell - 2008).

Por último, la capa más alta en la arquitectura está orientada a la interacción entre servicios. En ella se definen procesos de negocio, descubrimiento, agregación, etc. Así como la orquestación de servicios que consiste en la organización, coordinación, composición y monitorización de servicios web para conseguir un resultado derivado de la canalización correcta de las operaciones. La especificación más conocida de procesos de negocio es WS-BPEL de OASIS y WSCI (Web Service Choreography Interface) es un lenguaje de orquestación de servicios recomendado por W3C.

Esta solución de interoperabilidad y componentes software distribuidos ha tenido una gran repercusión en la programación de aplicaciones, sin embargo, los servicios web descritos en WSDL sólo disponen de una descripción a nivel sintáctico de su funcionalidad y parámetros. No disponen por tanto de ninguna definición del significado de dichos parámetros o mensajes. A lo sumo, en algunos casos se puede contar con comentarios o metadatos legibles solamente por humanos que facilitan la identificación de los servicios y su utilización para componer otros servicios.

Todos estos aspectos forman una plataforma completa de diversas capas que permite proporcionar servicios web, pero para realizar automáticamente los procesos de identificación, composición y ejecución de servicios web es necesario añadir información que permita hacer afirmaciones sobre los parámetros y operaciones que nos permitan conocer el tipo, relaciones de herencia, restricciones de cardinalidad, relaciones con otros elementos, etc. De esta manera, se podría utilizar un razonador que, dada una descripción semántica de un servicio buscado y una lista de descripciones de servicios existentes, pudiese identificar automáticamente qué servicios cumplen con el servicio buscado. Precisamente ése es el principal objetivo buscado al añadir semántica a los servicios web, incrementar la automatización de ciertas tareas que son llevadas a cabo con servicios, antes o durante la invocación. En el apartado 2.6 se comentarán en mayor detalle todos los aspectos relacionados con la semántica en los servicios web.

2.5.1 **UDDI**

Universal Description, Discovery and Integration en sus siglas en inglés, se trata de un registro basado en XML e independiente de la plataforma para listar los negocios existentes en Internet. Es una iniciativa industrial abierta promovida por OASIS para permitir la publicación de una lista de servicios de negocio para poder descubrirlos, usarlos y definir la forma en que los servicios o las aplicaciones software interactúan a través de la red. Por ello se podría definir como un catálogo de los negocios existentes en la red (Paolucci, Kawamura, Payne, & Sycara, 2002).

Un registro UDDI consta de tres componentes:

- Páginas blancas. Posee la dirección, contacto y los identificadores.
- Páginas amarillas. Contiene categorizaciones industriales basadas en taxonomías estándar.
- Páginas verdes. Información técnica sobre los servicios proporcionados por los negocios concretos.

UDDI fue propuesto como estándar de servicios web. Está diseñado para ser interrogado por mensajes SOAP y para proporcionar acceso a documentos WSDL (Web Services Description Language) describiendo el envoltorio del protocolo y los mensajes requeridos para interactuar con los servicios web listados en el directorio.

Para conocer más en detalle algunos datos acerca de UDDI, decir que fue escrito en Agosto de 2000 cuando sus autores se percataron de que los consumidores de servicios web deberían

encontrarse unidos con los suministradores de los mismos a través de un sistema broker ya fuese público o privado. En esta visión del mundo, cualquiera que necesitase un servicio iría al servicio del broker y seleccionaría uno capaz de proporcionarle el SOAP deseado o la interfaz de otro servicio que reuniese otro de los criterios establecidos. En ese mundo, el nodo público o el broker serían críticos para cualquiera. Para el consumidor, los brokers públicos o abiertos únicamente devolverían servicios listados para descubrimiento público por otros, mientras que para el productor del servicio, conseguir un buen posicionamiento en el índice de categorías sería crítico para el correcto desarrollo de su tarea.

UDDI se integró como un estándar en la Interoperabilidad de Servicios Web (WS-I), como un pilar central de la infraestructura de servicios web. A finales de 2005, era utilizado por más del setenta por ciento de las compañías de Fortune 500. Las especificaciones de UDDI apoyaban un acceso público Universal Business Registry en el que se construyó un sistema de nombres alrededor del servicio de broker de UDDI. IBM, Microsoft y SAP anunciaron que cerraban sus nodos UDDI público en enero de 2006. El lugar más común en el que un sistema de UDDI se puede encontrar está dentro de una empresa que lo utiliza para enlazar dinámicamente los sistemas del cliente con las implementaciones. Se diría que gran parte de la búsqueda de metadatos que permite UDDI no se utiliza para esta función relativamente simple. Sin embargo, el núcleo de la infraestructura comercial UDDI, cuando se despliegen en los Universal Business Registries, ha permitido mantener toda la información disponible para cualquier aplicación de cliente, con independencia de dominios informáticos heterogéneos.

Una vez conocida la información más relevante en cuanto al nacimiento, propósito y formación de UDDI, es momento de explicar el proceso a seguir para publicar servicios en UDDI. El primer paso consiste en determinar información básica sobre cómo definir la empresa y los servicios en UDDI. El siguiente paso, una vez determinada esta información, consiste en llevar a cabo el registro, ya sea mediante programación o a través de una interfaz de usuario basada en Web. Por último, se debe probar la entrada para asegurar que se registró correctamente y que aparece tal y como se esperaba en diferentes tipos de búsquedas y herramientas (Curbera et al., 2002).

1. **Definir la entrada de UDDI**

   Es necesario recopilar cierta información importante antes de establecer una entrada de UDDI.

   - Determinar los tModels (archivos WSDL) que utilizan las implementaciones del servicio Web. Al igual que sucede en el desarrollo de un componente COM, el servicio Web se ha desarrollado a partir de una interfaz existente o de una interfaz de diseño propio. En el caso de un servicio Web basado en una interfaz WSDL existente, se deberá determinar si el archivo WSDL se ha registrado en UDDI. Si es así, se deberá comprobar su nombre y tModelKey, que es el identificador GUID que generó UDDI cuando se produjo el registro. En caso contrario, se deberá crear un nuevo tModel para representar esta interfaz. El nombre de este tModel debería tener un formato URI (identificador de recursos uniforme), como MyCompany-com:SampleWebService-interface:v1, y señalar a la ubicación del archivo WSDL.
   - Si su servicio web es un servicio de Microsoft Visual Studio .NET, se podrá generar una descripción WSDL utilizando una cadena de consulta desde el archivo ASMX. No

obstante, el archivo WSDL generado por Visual Studio .NET se relaciona estrechamente con el punto de acceso para la invocación del servicio Web, lo cual puede no resultar adecuado cuando la interfaz del servicio tiene varias implementaciones. Esto no supondrá ningún problema si la intención es que el archivo WSDL sólo tenga una implementación.

- Determinar el nombre de la empresa y una breve descripción de la misma en varios idiomas, si es necesario, así como los contactos principales para los servicios Web que ofrece. UDDI es compatible con el espacio de nombre xml:lang, lo que permite a las empresas ofrecer su descripción en varios idiomas. Asimismo, UDDI permite enumerar los contactos, incluyendo datos como el correo electrónico, el teléfono y la dirección. Esta lista de contactos muestra los recursos de una empresa con los que se puede poner en contacto en relación con los servicios web ofrecidos. Por ejemplo, si un usuario desea comenzar a utilizar el servicio web deberá ponerse en contacto con el responsable de relaciones comerciales correspondiente pero, ¿cómo puede llegar a saber quién es? ¿Existe algún contacto para obtener asistencia técnica a la hora de utilizar los servicios web de la empresa? También se debería incluir en la lista a esta persona.
- Determinar las categorías e identificaciones adecuadas para la empresa. Se podrá explorar los sistemas taxonómicos compatibles con UDDI actualmente en el nodo Microsoft UDDI (http://uddi.microsoft.com/default.aspx). Estos sistemas son, por el momento, North American Industry Classification System (NAICS), Universal Standard Products and Services Codes (UNSPSC), ISO 3166, Standard Industry Classification (SIC) y GeoWeb Geographic Classification. Se deben seleccionar las categorías que representan de forma más acertada a la empresa en cuestión.
- Determinar los servicios Web que la empresa ofrece a través de UDDI. A continuación, se deberán determinar los servicios web que desea registrar la empresa en el nodo público UDDI. ¿Existen varios puntos de acceso para este servicio? ¿Es preciso que los clientes conozcan otros parámetros y otra información para utilizar el servicio web? Resulta importante destacar que no todo el mundo puede obtener acceso a un servicio Web porque éste se haya registrado en UDDI. A una entrada de registro UDDI le pueden acompañar medidas de seguridad, autorización y autenticación. No basta que el usuario sepa que existe un servicio Web para que pueda invocarlo. Puede existir una comunicación fuera de banda entre empresas antes de permitir el acceso a un servicio web.
- Determinar las categorías adecuadas para los servicios. Los servicios web se pueden categorizar del mismo modo que las empresas. No obstante, una empresa se debe categorizar a nivel empresarial, como por ejemplo NAICS: Software Publisher (51121), y el servicio web (de reserva hotelera, en este caso) se debería categorizar en el nivel de servicios, como NAICS: Hotels and Motels (72111).

2. Registrar la entrada de UDDI

Una vez finalizada la tarea de definición, el siguiente paso consiste en registrar la empresa. Se deberá obtener una cuenta con un registro UDDI. Esta operación no se puede realizar mediante programación, ya que deberá mostrar su conformidad con una declaración de

condiciones de uso. El nodo de Microsoft utiliza Passport para la autenticación, así que deberá adquirir una cuenta de Passport para continuar con el registro.

En este punto se ofrecen dos opciones: utilizar la interfaz de usuario Web del nodo de Microsoft o realizar el registro mediante programación dirigiendo al propio nodo las llamadas a API de SOAP. Si no se piensa modificar la entrada o ésta es relativamente simple, bastará con la interfaz de usuario Web. No obstante, si se pretende actualizar la entrada con frecuencia, o bien, ésta es más compleja, resulta recomendable realizar el proceso de registro con secuencias de comandos, utilizando el SDK de Microsoft UDDI. Además, la interfaz de usuario de Microsoft no está localizada en otros idiomas, así que se deberá registrar mediante programación para disfrutar esa característica de la API de UDDI.

3. Buscar la entrada en UDDI

Es recomendable realizar tres comprobaciones una vez registrada la entrada en UDDI. En primer lugar, utilizando la interfaz de usuario Web de Microsoft, se debe buscar la empresa por su nombre y categorizaciones para verla entre los conjuntos de resultados devueltos. En segundo lugar, abrir Visual Studio .NET y asegurar que aparece en el cuadro de diálogo "Agregar referencia Web". Si no aparece, se puede deber a que el tModel no se categorizó correctamente utilizando la taxonomía uddi-org:types descrita anteriormente. Se podrá agregar el servicio Web al proyecto y generar el código proxy basado en el archivo WSDL. Por último, transcurridas 24 horas, la entrada se replicará al nodo de IBM y podrá buscarla con su IU[1].

Por último comentar que UDDI y WSDL funcionan como especificaciones gratuitas que facilitan el desarrollo de una colección de software basado en servicios web. WSDL ofrece un modo formal de definir servicios Web, independientemente del proveedor, que permitirá realizar llamadas a procedimientos remotos de próxima generación, mientras que UDDI proporciona una amplia infraestructura estandarizada que permite al usuario describir y descubrir servicios web. Mediante la combinación de estos dos estándares, se podrá desarrollar todo un universo de servicios web.

2.5.2 **WSDL**

Web Services Description Language, es un lenguaje de descripción de servicios en su versión 2.0. Proporciona un modelo y un formato XML para describir servicios web (Booth & Liu, 2007). En particular, WSDL 2.0 permite la separación de la descripción de la funcionalidad abstracta ofrecida por el servicio de los detalles más concretos relativos al ''cómo'' y ''dónde'' obtener dicha funcionalidad. Previas a la versión 2.0, que fue la que llegó a convertirse en una recomendación del W3C, existieron otras versiones que evolucionaron desde la 1.0 en el 2000 hasta la 1.2 en el año 2003.

[1] https://www-3.ibm.com/services/uddi/protect/find

Un documento WSDL define los servicios como una colección de puertos. En WSDL, la definición abstracta de endpoints y mensajes se encuentra separada de la red concreta en la que se encuentran o el formato de los datos vinculados (''bindings''). Esto permite la reutilización de definiciones abstractas: mensajes, que son descripciones abstractas de los datos intercambiados, y tipos de puerto que son colecciones abstractas de operaciones. El protocolo concreto y las especificaciones del formato de los datos para un tipo de puerto concreto constituyen una vinculación reutilizable. Un puerto se define mediante la asociación de una dirección de red con un binding reutilizable. Una colección de puerto define un servicio. De hecho, un documento WSDL utiliza los siguientes elementos en la definición de los servicios de su red:

- Types, es un contenedor para definición de tipos de datos mediante el uso de algún sistema de tipos (por ejemplo, XSD).
- Message, es una definición abstracta y tipada de los datos que están siendo comunicados.
- Operation, descripción abstracta de una acción soportada por el servicio.
- Port Type, un conjunto abstracto de operaciones soportadas por uno o más endpoints.
- Binding, protocolo concreto y especificación del formato de datos para un tipo de puerto particular.
- Port, un único endpoint definido como una combinación de binding y una dirección de red.
- Service, colección de endpoints relacionados.

Es importante comprobar que WSDL no introduce un nuevo lenguaje de definición de tipos. WSDL reconoce la necesidad de sistemas de tipos enriquecidos para la descripción de formatos de mensajes y soporte de especificaciones XML Schema. Aunque, no es razonable esperar una única gramática de tipos para describir todos los formatos de mensajes presentes y futuros, WSDL permite usar otro tipo de lenguajes de definición mediante extensibilidad (*Web Services Description Language (WSDL) 1.1, W3C Note*, 2001).

Además. WSDL define un mecanismo de vínculo común. Se utiliza para agregar un protocolo específico, formato de datos o estructura a un mensaje abstracto, operación o endpoint. Permite la reutilización de definiciones abstractas.

Por tanto, una descripción WSDL está estructurada en dos etapas fundamentales:

1. En el nivel abstracto, WSDL 2.0 describe un servicio web en términos de los mensajes que éste envía y recibe. Los mensajes están descritos independientemente del formato utilizado, empleándose para ello un sistema de tipos como XML Schema. Una operación (''operation'') asocia un patrón de intercambio de mensajes a uno o más mensajes. Un patrón de intercambio de mensajes (''message exchange pattern'') identifica la secuencia y cardinalidad de los mensajes enviados y/o recibidos, así como a quién se envían y/o de quién se reciben. Una interfaz (''interface'') agrupa las operaciones sin concretar un formato específico.
2. En el nivel concreto, una vinculación (''binding'') especifica los detalles relativos al formato para una o más interfaces. Por su parte, un punto final (''endpoint'') asocia

una dirección de red con una vinculación. Finalmente, un servicio (''service'') agrupa los puntos finales que implementan una interfaz común.

El modelo conceptual de WSDL 2.0 puede verse como un conjunto de componentes a los que se les asocian propiedades y que, colectivamente, describen un Servicio Web. Este modelo es lo que se denomina ''Modelo de Componentes'' de WSDL 2.0 y que se puede observar a continuación:

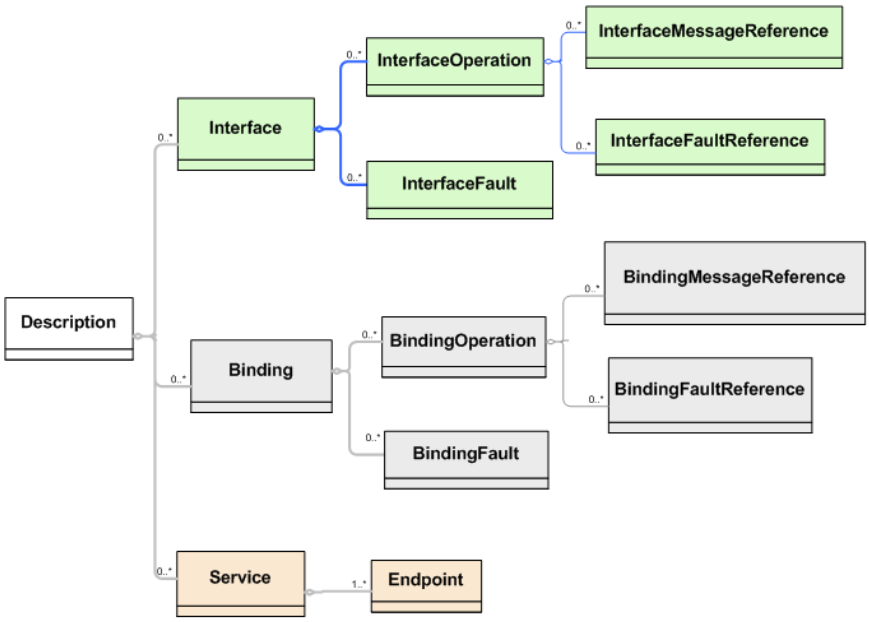


Figura 16. Modelo de componentes de WSDL

Descripción (''description'') es el componente contenedor dentro del cual toda la información relativa a un servicio debe incluirse. El componente descripción contiene dos categorías de componentes, los componentes de tipos del sistema (declaración de elementos y definiciones de tipos) y los componentes WSDL 2.0 (interfaces, vinculaciones y servicios). Como se puede comprobar parcialmente en la figura anterior, ''interface'', ''binding'', ''service'', ''element declaration'' y ''type definition'' son los componentes que están directamente contenidos en ''description'', estos son los denominados top-level components. Estos componentes pueden a su vez contener otros componentes, denominados componentes anidados.

WSDL se usa a menudo en combinación con SOAP y XML Schema. Un programa cliente que se conecta a un servicio web puede leer el WSDL para determinar qué funciones están disponibles en el servidor. Los tipos de datos especiales se incluyen en el archivo WSDL en forma de XML Schema. El cliente puede usar SOAP para hacer la llamada a una de las funciones listadas en el WSDL.

Otra de las descripciones es el estándar SA-WSDL ("Semantic Annotations WSDL"), su principal objetivo es resolver la posible ambigüedad en la descripción de los Servicios Web que puede derivar de la especificación WSDL 2.0. Esta especificación no incluye semántica en la

descripción de los servicios de modo que es posible que dos servicios con significados distintos tengan una descripción similar.

2.6 Servicios Web Semánticos

Una vez conocidos los aspectos más importantes de los servicios web, se puede definir un servicio web semántico como la conjunción de un servicio web y una anotación semántica sobre dicho servicio para crear servicios web ''inteligentes'' (Martin et al., 2004). Una anotación o descripción semántica consiste en asociar conceptos y relaciones de una ontología con parámetros y operaciones de un servicio web.

Los servicios web semánticos son una línea importante de la Web Semántica, que propone, por tanto, describir no sólo información, sino definir ontologías de funcionalidad y procedimientos para describir servicios web: sus entradas y salidas, las condiciones necesarias para que se puedan ejecutar, los efectos que producen, o los pasos a seguir cuando se trata de un servicio compuesto. Estas descripciones procesables por máquinas permitirían automatizar el descubrimiento, la composición, y la ejecución de servicios, así como la comunicación entre unos y otros.

La aplicación de este tipo de servicios semánticos permite la creación de sistemas ''inteligentes'' capaces de extraer la información necesaria en cada momento y localizar los objetivos de forma sencilla y automática, suponiendo un gran avance y un ahorro en tiempo y costes. Además, estos servicios facilitan la interoperabilidad para así aprovechar todo su potencial. La Figura 17 muestra los niveles en los que cada tecnología ataca el problema de la interoperabilidad y los recursos empleados para ello.

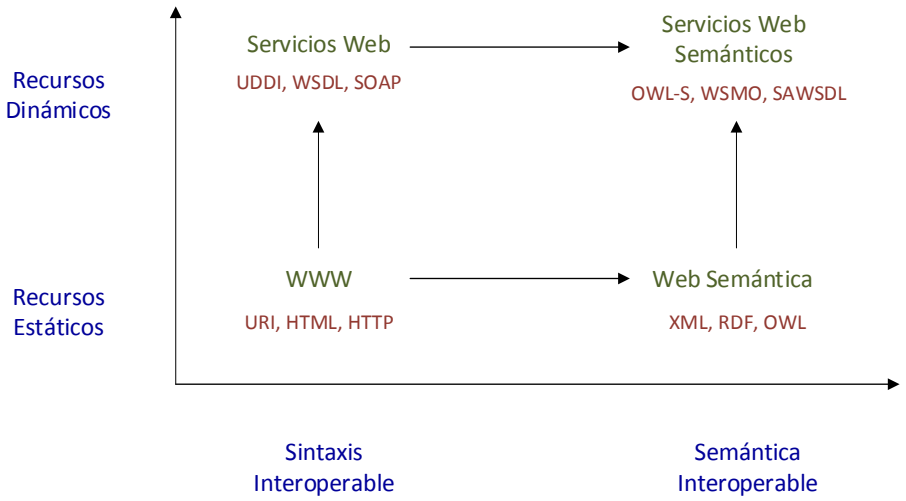


Figura 17. Niveles de interoperabilidad de servicios

Para ilustrar mejor toda esta explicación supóngase la existencia de un servicio web que ofrece un servicio de compra de libros, revistas y material audiovisual. Ese servicio dispondrá de una operación ''buscar'' que tendrá al menos un parámetro que será el título o autor de la obra que se desea encontrar. Esta operación, desde un punto de vista sintáctico, podría ser confundida con búsquedas de otros artículos o simplemente no identificada como tal.

Por otro lado, si se conociese la existencia de una ontología, que describiese el entorno en el que se encuentran los elementos libro, revista, artículo, etc. cada uno de ellos con sus propiedades. Partiendo del servicio web y de la ontología descrita, se puede crear una descripción de servicio web que anota los parámetros del servicio de búsqueda de libros para restringirlos a los elementos y relaciones que deben satisfacer. Así, cualquier localizador automático de servicios podría identificar claramente si el servicio le ofrece lo que él necesita o no, en base a sus entradas y salidas, por ejemplo.

El W3C (World Wide Web Consortium) está examinando diversas propuestas para la anotación semántica de los servicios con el objetivo de establecer un estándar para la tecnología de servicios web semánticos: OWL-S (Elenius et al., 2005), WSMO (Roman et al., 2005), SWSF (2005), WSDL-S (Li et al., 2006) y SAWSDL (Verma & Sheth, 2007).

OWL-S, WSMO y SWSF siguen el mismo esquema, definir ontologías que sirvan de marco conceptual para determinar los conceptos y propiedades que son necesarios para definir un servicio Web. Por otro lado, WSDL-S y SAWSDL toman un camino diferente. En estas propuestas se prefiere elaborar un modelo de extensión sobre WSDL que permita asociar anotaciones semánticas a documentos WSDL mediante elementos de extensibilidad.

Junto a estas propuestas al W3C, se han desarrollado herramientas que integran en una aplicación todo el ciclo de vida de los servicios web semánticos.

Ejemplos de estas aplicaciones son WSMX (Zaremba & Vitvar, 2008) e IRS-III (Domingue et al., 2008):

- WSMX es un entorno de ejecución que permite realizar el descubrimiento, selección, mediación e invocación dinámicos de servicios web semánticos.
- El sistema IRS es un marco de trabajo que permite a las aplicaciones describir semánticamente y ejecutar servicios Web. Esta aplicación incluye, además, servicios para el razonamiento semántico en el contexto de la Web Semántica.

Existen, del mismo modo, herramientas que pretenden facilitar la tarea del desarrollador de servicios web semánticos. Un ejemplo destacado dentro de este grupo de aplicaciones es el ''Web Service Modeling Toolkit'' (WSMT) (Kerrigan, Mocan, Tanler, & Fensel, 2007). El WSMT es un entorno de desarrollo integrado para servicios web semánticos que permite a los implementadores desarrollar tanto servicios web, como ontologías, objetivos y mediadores a través del formalismo WSMO, un modelo conceptual para la descripción de varios aspectos relacionados con los servicios web semánticos que toma el WSMF ("Web Service Modelling Framework") como punto de partida redefiniendo y extendiendo sus conceptos. Esta herramienta ayuda al desarrollador a lo largo de todo el ciclo de vida del desarrollo software relacionado con los servicios web semánticos: requisitos, diseño, implementación, test, etc. Sin embargo, la mayor parte de las herramientas se centra en el proceso que no está cubierto por los entornos de desarrollo de aplicaciones actuales: la anotación semántica del servicio web.

Así, por ejemplo, la aplicación ASSAM (Hess, Johnston, & Kushmerick, 2004) utiliza un algoritmo de machine learning para sugerir al usuario el elemento ontológico que debe utilizar para anotar un elemento del fichero WSDL. Para esto, se tienen en cuenta las anotaciones realizadas previamente sobre servicios similares. Otra herramienta para la anotación semántica, en este caso totalmente manual, de servicios Web es el ''OWL-S Editor'' (Elenius et al., 2005). Esta herramienta ha sido implementada como un plugin de Protege que, a partir de un fichero WSDL y un conjunto de ontologías cargadas en Protege, permite al usuario asociar elementos de estas ontologías a los distintos apartados de WSDL, generando como salida la descripción en OWL-S del servicio. Otro ''OWL-S Editor'', desarrollado por James Scicluna en la Universidad de Malta, funciona como una aplicación de escritorio independiente con una funcionalidad similar.

En cuanto a la ejecución de servicios web semánticos cabe destacar tres plataformas que facilitan el manejo de este tipo de servicios: INFRAWEBS, WSMX e IRS-III

1. INFRAWEBS es un proyecto FP6 a gran escala que comenzó en agosto de 2004 con una duración de 30 meses. El principal objetivo de INFRAWEBS es el desarrollo de un conjunto de herramientas orientadas a aplicación para la creación, el mantenimiento y la ejecución de servicios web semánticos basados en WSMO durante su ciclo de vida completo. La plataforma INFRAWEBS contiene dos elementos principales:
   - Un gestor de conocimiento basado en una memoria organizativa
   - Un cliente para el lanzamiento y seguimiento de SWS

2. WSMX es una plataforma que se encuentra actualmente en desarrollo por el grupo de trabajo SDK de Deri Internacional. WSMX tiene como objetivo principal el proporcionar una plataforma de ejecución que sea capaz de interpretar objetivos de petición de servicios para:
   - Descubrimiento de servicios relacionados
   - Seleccionar el servicio más apropiado
   - Proveer mediación de datos en caso necesario
   - Realizar invocación de servicios

   Para esto WSMX está basado en el modelo conceptual proporcionado por WSMO.

3. La plataforma IRS-III es una infraestructura para la publicación, el descubrimiento, la ejecución y la composición de servicios web semánticos.

Al igual que sucediese en el caso de WSMX, esta plataforma se basa en el modelo conceptual de WSMO. Fue desarrollado en el Knowledge Media Institute de la Open University (UK) como parte del proyecto DIP. IRS-III proporciona una plataforma e infraestructura para la creación de servicios web semánticos basados en WSMO, construida a partir de la implementación anterior IRS-II.

2.6.1 **OWL-S**

OWL-S (Web Ontology Language for Services) es una ontología de servicios web basada en OWL y desarrollada por la rama de servicios web semánticos del programa DAML. Se trata de una evolución de DAML-S y proporciona un conjunto esencial de constructores de lenguaje de marcado para describir las propiedades y capacidades de los servicios web de una forma inequívoca e interpretable por las máquinas (Martin et al., 2004). Ofrece un entorno adecuado para describir servicios desde varias perspectivas, más precisamente, las ontologías de alto nivel caracterizan servicios a través de las sub-ontologías: ''Service Profile'' (para la publicación y el descubrimiento de servicios), ''Service Model'' (para la descripción del servicio) y ''Service Grounding'' (para la invocación de servicios). Se le puede denominar como lenguaje, ya que proporciona un vocabulario estándar que puede ser usado junto los otros aspectos del lenguaje de descripción OWL. El marcado de los servicios web mediante OWL-S facilita la automatización de las tareas de los mismos incluyendo el descubrimiento de servicios, ejecución, interoperabilidad, composición y monitorización (Burstein et al., 2004). De esta forma se proporcionan respuestas a preguntas esenciales como qué ofrece el servicio, cómo funciona el servicio y cómo se interactúa con él.

Para reflejar las características y capacidades que posee OWL-S se presenta el siguiente ejemplo. Supóngase que una persona entra en una agencia de viajes e introduce sus fechas e información de destino en una aplicación que utilice OWL-S, eligiendo los siguientes criterios:

- Viaje sin escalas y tarifa más barata de vuelo
- Disponibilidad y menor precio por día para el alquiler de un coche.

En una aplicación con soporte OWL-S y sin ningún tipo de intervención de la persona la aplicación se encargaría de lo siguiente:

1. Descubrir el conjunto de servicios de la aerolínea que proporciona las tarifas para viajar al destino seleccionado (descubrimiento automático de servicios).
2. Obtener las tarifas de los servicios descubiertos de la aerolínea y seleccionar aquella que cumpla los requisitos de vuelo directo y menor precio.
3. Descubrir un conjunto de servicios que ofrecen alquiler de coches en el destino seleccionado.
4. Consultar la disponibilidad y tarifas y seleccionar aquella con menor precio diario teniendo en cuenta las fechas del viaje (se corresponde con composición automática e invocación de servicios, a salida del servicio de viaje se transforma en entrada del servicio de alquiler de coches).
5. Mostrar al usuario el itinerario y características y solicitar confirmación.

Como se puede comprobar, el ejemplo deja bastante claras las principales ideas que se pretenden conseguir mediante el desarrollo y aplicación de OWL-S. A continuación se proporcionará información más detallada acerca de las diversas características y enfoques que otorga esta ontología.

2.6.1.1 Motivaciones

En OWL-S se tienen en cuenta tanto los servicios simples o ''atómicos'' como los complejos o ''compuestos''. Los servicios atómicos son aquellos en los que un sólo programa accesible desde la web, un sensor o un dispositivo es invocado por un mensaje, realizar su tarea y puede producir una respuesta. Con los servicios atómicos no existe una interacción intermedia entre el usuario y el servicio. Por otro lado, los servicios complejos se encuentran compuestos por multitud de servicios primitivos y pueden necesitar una interacción o conversación entre aquél que solicita y los servicios que se están utilizando. Una vez comentados estos aspectos, se muestran a continuación los tres tipos de tareas que permite ejecutar OWL-S:

1. **Descubrimiento automático de servicios**. Se trata de un proceso automático para la localización de servicios web que pueden proporcionar una ser de capacidades concretas que se ajusten a los requisitos especificados por el cliente. Con los servicios de marcado de OWL-S la información necesaria para el descubrimiento de servicios puede ser especificada como anotaciones semánticas interpretables por el ordenador en los servicios de la web y un registro de servicios o motores de búsqueda basados en ontologías que pueden ser empleados para localizar los servicios automáticamente. Alternativamente, un servidor puede anunciarse proactivamente a sí mismo en OWL-S con un registro de servicio de forma que aquellos que busquen puedan encontrarle cuando consultan en el registro. Por ello, OWL-S habilita anuncios declarativos de las propiedades y capacidades del servicio que pueden ser usados para el descubrimiento automático de servicios.
2. **Invocación automática de servicios web**. Este concepto se refiere a la invocación automática de un servicio web por parte de un programa o agente, dada sólo una descripción declarativa del servicio, al contrario de lo que ocurre cuando se preprograma un agente para invocar un servicio concreto. La ejecución de un servicio web puede verse como una colección de llamadas remotas a procesos. Las marcas OWL-S de los servicios web proporcionan una API interpretable por la máquina que incluye la semántica de los argumentos para ser especificada cuando se ejecutan esas llamadas, y la semántica de ellas se devuelve en mensajes cuando el servicio se completa o falla. Un agente software debe ser capaz de interpretar esas marcas para entender qué entrada es necesaria para invocar el servicio y que información debe devolver. OWL-S, conjuntamente con ontologías específicas del dominio en OWL, proporciona significados estándar para especificación declarativa de APIs para servicios web que habilitan este tipo de ejecución automática de servicios web.
3. **Composición y ejecución automática de servicios web**. Esta tarea involucra la selección, composición e interoperación automática de servicios web para realizar tareas complejas dada una descripción de alto nivel de un objetivo. Con las marcas OWL-S, la información necesaria para seleccionar y componer los servicios web se encuentra codificada en los sitios web. El software puede escribirse para manipular estas representaciones junto con una especificación de los objetivos de la tarea, para alcanzar la tarea de forma automática. Para soportar esto, OWL-S proporciona especificaciones declarativas de los prerrequisitos y consecuencias de

aplicación de servicios individuales y un lenguaje para describir la composición de servicios y la interacción de los flujos.

Cualquier programa, sensor o dispositivo accesible desde la web que sea declarado como servicio puede ser requerido como servicio. OWL-S no se opone a declarar páginas estáticas y simples como servicios. Pero la principal motivación en la definición de OWL-S es la de soportar tareas de mayor complejidad como se han descrito previamente.

2.6.1.2 Ontologías OWL-S

En su forma más simple, una ontología OWL-S define elementos que describen la interfaz que proporciona un servicio al ''mundo exterior''. Como queda especificado en WSDL, las entradas y salidas de un servicio se mapean con las clasificaciones de una ontología OWL. Haciendo eso, cualquier aplicación capaz de comprender los conceptos presentes puede utilizar sus respectivas ontologías para usar el servicio sin necesidad de especificar conocimiento de su interfaz de servicio.

2.6.1.3 OWL-S Description Elements

Un servicio en OWL-S se describe mediante tres elementos:

1. Service Profile, describe lo que hace el servicio. Explica que servicios lleva a cabo, detalla las limitaciones de aplicación y calidad del servicio y especifica los requisitos que debe satisfacer la entidad que solicita para que el proceso se produzca correctamente. Esta información es utilizada por los consumidores durante el descubrimiento del servicio.
   Además de la información puramente identificativa, en el Service Profile se describe funcionalmente el servicio en base a las entradas, salidas, precondiciones y efectos, comúnmente conocidos como IOPE's (Inputs, Outputs, Preconditions y Effects). Las entradas y salidas se representan por las propiedades hasInput y hasOutput respectivamente y hacen referencia a las entradas y salidas que se definen posteriormente en el Service Model. Un servicio define además una serie de condiciones que deben cumplirse previamente a su ejecución, así como las postcondiciones resultado de la ejecución del servicio. Las primeras se definen con la propiedad hasPrecondition, que hace referencia a una precondición definida en el Service Model y las segundas se definen con la propiedad hasResult que especifica las condiciones de los parámetros de salida.
2. Service Model, describe como se usa el servicio. Detalla la semántica contenida en las consultas, las condiciones bajo las que determinadas salidas se producen y, cuando sea necesario, procesos guiados paso a paso para alcanzar dichas salidas.
   Cualquier servicio es visto como un proceso, bien sea atómico o compuesto, que bajo ciertas condiciones y unos parámetros de entrada, produce una serie de salidas que cumplen ciertas precondiciones. Este enfoque basado en procesos se apoya sobre

trabajos ya existentes y de referencia en materia de inteligencia artificial, lenguajes de programación, sistemas distribuidos y estándares en el campo de la definición de flujos de trabajo.

El modelo de un servicio puede ser utilizado por los agentes de búsqueda hasta de cuatro formas distintas:

- Para analizar con más detalle si el servicio cumple los requisitos buscados.
- Para componer descripciones de múltiples servicios.
- Para coordinar clientes durante la ejecución de los servicios.
- Para monitorizar la ejecución del servicio.

Como se describió al presentar el Service Profile, es en el Service Model donde se definen las instancias para los parámetros de entrada, salida, precondiciones y resultados. Los parámetros de entrada y salida son representados como variables de SWRL (Semantic Web Rule Language) y en cada uno de ellos se define el tipo de valores que puede instanciar. Las precondiciones y resultados (o efectos) son tratados como expresiones lógicas que funcionan a modo de literales que pueden estar descritos en un lenguaje basado en XML (SWRL) o en otros lenguajes.

3. Service Grounding, especifica los detalles de cómo acceder o invocar al servicio. Incluye protocolos de comunicación, formato de mensajes, técnicas de serialización y transformaciones para cada entrada y salida, así como otros detalles específicos del servicio como número de puertos usados al contactar con el servicio.

   En el Service Grounding de un servicio los mensajes intercambiados con dicho servicio se describen en el propio lenguaje de la implementación del servicio. OWL-S no añade ninguna restricción sobre el lenguaje en el que esté implementado el servicio.

2.6.1.4 OWL-S Discovery and Execution Elements

Por sí solo, OWL-S es un lenguaje para el marcado de servicios web. Llega a ser útil cuando se combina con herramientas capaces de explotar los servicios web descritos empleando OWL-S. Uno de los ejemplos más conocidos de herramienta es CODE (CMU's OWL-S Development Environment) (Srinivasan, Paolucci, & Sycara, 2005). CODE soporta el desarrollo completo de servicios web en OWL-S desde la generación de las descripciones OWL-S (desde código Java a documentos WSDL) hasta el despliegue y registro del servicio.

Además de lo que ofrecen otras herramientas para la descripción de servicios, CODE incluye el OWL-S Matchmaker y el OWL-S Virtual Machine (VM). El primero sirve como catálogo de servicios definidos usando OWL-S. Los proveedores del servicio registran las descripciones OWL-S de los servicios con el OWL-S Matchmaker. Las aplicaciones cliente pueden realizar consultas al OWL-S Matchmaker con una descripción ontológica de las entradas y salidas deseadas. The OWL-S Matchmaker mapea la consulta con el catálogo de servicio y devuelve una lista ordenada de servicios en función del más cercano a la consulta. Por otro lado la OWL-S VM se usa para invocar servicios empleando OWL-S. Después de que la aplicación cliente selecciona un servicio de la lista ordena, formula su consulta utilizando el formato especificado por la ontología OWL y la envía a la OWL-S VM. Utilizando el XSLT presente en el Service Grounding, la OWL-S VM formatea la consulta para que coincida con el formato requerido por el servicio. Entonces, se invoca el servicio en nombre del cliente. Cuando recibe una respuesta,

la OWL-S VM usa otro XSLT en el Service Grounding para volver a formatear la respuesta para que sea apropiado respecto al esperado por la ontología. Finalmente, la OWL-S VM envía la respuesta de vuelta a la aplicación cliente. De esta manera, la aplicación del cliente no necesita conocer nada acerca de la forma de interactuar con el servicio, la OWL-S VM actúa como mediador para la pregunta y la respuesta.

Ambos, los elementos del OWL-S Matchmaker y de la OWL-S VM poseen una API para aplicaciones Java para el descubrimiento y la invocación de servicios.

También existen otras herramientas como ''OWL-S Modeller'' (Navas-Delgado, Kerzazi, & Aldana-Montes, 2004), que se trata de una extensión de WSMO Studio para permitir el diseño de anotaciones de servicios mediante OWL-S, que presenta una forma de que los desarrolladores de servicios Web Semánticos puedan hacer uso de un mismo entorno para producir servicios anotados con ambas propuestas.

2.6.2 **WSMO**

WSMO identifica cuatro elementos de alto nivel como modelo de conocimiento para describir los servicios web semánticos: ontologías, objetivos, servicios web y mediadores.

La propuesta relacionada con la descripción semántica de los servicios web que cronológicamente siguió a OWL-S fue WSMO (Web Service Modeling Ontology) en 2005 (Roman et al., 2005). Con una visión similar a la de los autores de OWL-S, WSMO proporciona un marco de trabajo conceptual y un lenguaje formal para describir de forma semántica todos y cada uno de los aspectos relevantes relacionados con los servicios web para, así, facilitar la automatización de tareas tales como el descubrimiento, la combinación y la invocación de servicios electrónicos sobre la web. En WSMO, un servicio web se define como una entidad computacional capaz, una vez invocada, de satisfacer el objetivo de un usuario.

WSMO está basado en el modelo conceptual propuesto en el "Web Service Modeling Framework" (WSMF) (Dieter Fensel & Bussler, 2002), que identifica cuatro elementos de alto nivel como modelo de conocimiento para describir los servicios web semánticos: ontologías, objetivos, Servicios Web y mediadores:

1. **Ontologías**: proporcionan la terminología que será usada por los restantes elementos.
   En WSMO, las ontologías son la clave para conectar la semántica conceptual del mundo real definida y acordada por comunidades de usuarios. Con este propósito, las ontologías definen una terminología común concertada indicando conceptos, relaciones entre conceptos y axiomas, que se pueden definir como expresiones en algún lenguaje lógico. Los axiomas se utilizan para capturar las propiedades semánticas de las relaciones y los conceptos. Por tanto, las ontologías proporcionan en WSMO la terminología básica de la que harán uso los restantes elementos para describir los aspectos relevantes del dominio del discurso.
   En WSMO, las ontologías son la clave para conectar la semántica conceptual del mundo real definida y acordada por comunidades de usuarios. Con este propósito, las ontologías definen una terminología común concertada indicando conceptos, relaciones entre conceptos y axiomas, que se pueden definir como expresiones en

algún lenguaje lógico. Los axiomas se utilizan para capturar las propiedades semánticas de las relaciones y los conceptos. Por tanto, las ontologías proporcionan en WSMO la terminología básica de la que harán uso los restantes elementos para describir los aspectos relevantes del dominio del discurso.

2. **Objetivos**: representan los deseos de los usuarios o intenciones que deben ser satisfechas por algún servicio web.

   Un objetivo se refiere a la representación de una meta para cuya consecución es necesaria la ejecución de un servicio web. Los objetivos pueden ser descripciones de servicios web que, potencialmente, satisfarían los deseos del usuario. De forma similar a lo que ocurre en las descripciones de servicios web, las ontologías pueden utilizarse para definir la terminología del dominio que describe los aspectos relevantes de los objetivos.

3. **Descripciones de servicios web**: define los aspectos funcionales y de comportamiento de un servicio web.

   Un servicio web en WSMO describe a una entidad computacional que permite el acceso a servicios que proporcionan algún valor en un dominio. La descripción de un servicio web en WSMO consiste en aspectos funcionales, no funcionales y de comportamiento del servicio. Esta descripción comprende las capacidades, interfaces y el funcionamiento interno del servicio. Todos estos aspectos de un servicio Web son descritos mediante el uso de la terminología definida por parte de las ontologías. Se distingue, además, el concepto de servicio web, entendido como entidad computacional capaz de satisfacer un objetivo de usuario, del concepto servicio, entendido como el valor real proporcionado por la invocación del servicio web (un mismo servicio web puede proveer varios servicios).

4. **Mediadores**: tienen el propósito de gestionar de forma automática los problemas de interoperabilidad que surjan entre los restantes elementos.

   Finalmente, por mediadores se entienden aquellos elementos capaces de superar los problemas de interoperabilidad que puedan surgir entre los distintos elementos de WSMO. Estos son el concepto clave para resolver incompatibilidades a nivel tanto de datos, como de procesos y protocolos. La mediación de datos permite resolver incongruencias entre diferentes terminologías utilizadas. Por su parte, la mediación a nivel de protocolo trata con los problemas relacionados con las incompatibilidades en la comunicación entre servicios web. Por último, la mediación a nivel de procesos permite la combinación de servicios web (y objetivos).

   Para tratar todas estas cuestiones, se distinguen cuatro tipos de mediadores:

   i. ggMediators: mediadores que conectan dos objetivos. Esta conexión representa el refinamiento del objetivo fuente en el objetivo destino, o establece una equivalencia si ambos objetivos son intercambiables.
   ii. ooMediators: mediadores que importan ontologías y resuelven posibles incompatibilidades en la representación de las ontologías.
   iii. wgMediators: mediadores que conectan servicios web con objetivos. La conexión entre un servicio web y un objetivo significa que el servicio web (total o parcialmente) satisface el objetivo asociado.
   iv. wwMediators: mediadores que conectan dos servicios web.

Teniendo como base los conceptos identificados en WSMF, WSMO proporciona una especificación ontológica para los elementos que conforman el núcleo de los servicios web semánticos. A diferencia de OWL-S, el lenguaje de ontologías utilizado con este propósito en WSMO es WSML (De et al., 2005).

En la Figura 18 se muestran los conceptos de alto nivel que constituyen la ontología de WSMO:

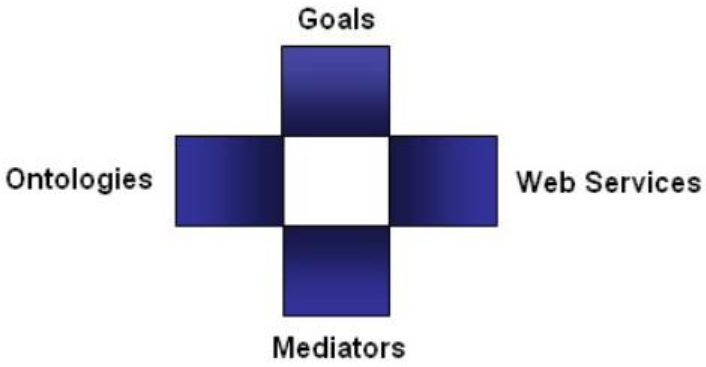


Figura 18. Elementos de alto nivel de WSMO

Todos los conceptos de nivel superior comentados son elementos WSMO y poseen una serie de propiedades no funcionales que los caracterizan de uno u otro modo. La Figura 19 muestra las relaciones existentes entre dichos conceptos:

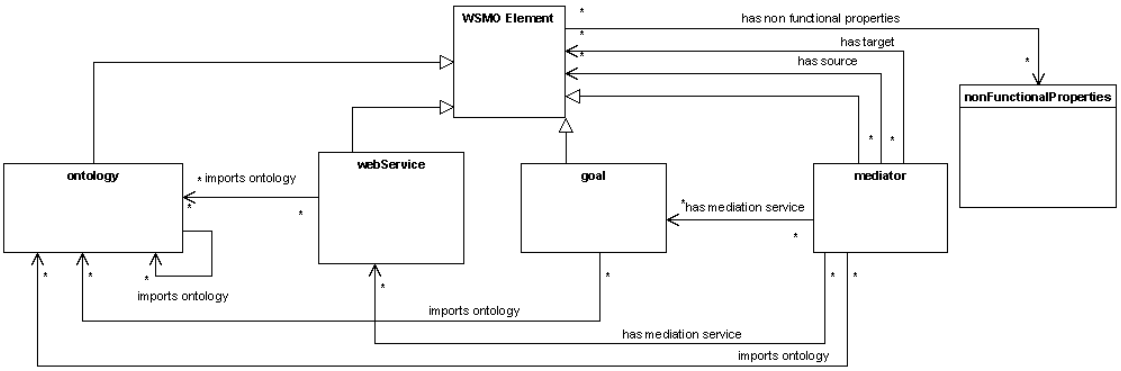


Figura 19. Relaciones entre conceptos de alto nivel de la ontología de WSMO

2.6.3 OWL-S vs WSMO

Tanto OWL-S como WSMO tienen un propósito común, la especificación de información semántica relacionada con los servicios web para permitir la automatización de tareas como el descubrimiento, la composición y la ejecución. Sin embargo, existen diferencias sustanciales entre ambas aproximaciones (Bruijn, Lara, Polleres, & Fensel, 2005). Entre las diferencias más significativas, cabe destacar las siguientes:

- OWL-S no separa lo que un usuario quiere de lo que un servicio puede proporcionar (diferencia entre objetivo y descripción de servicio web en WSMO)
- WSMO permite indicar propiedades no funcionales en todos y cada uno de los elementos de la ontología, haciendo, además, uso de vocabularios comúnmente aceptados como "Dublin Core" (DC) y "Friend of a Friend" (FOAF), mientras que OWL-S sólo permite indicar estas propiedades en la ontología del perfil y no se basa en especificaciones de metadatos estándar
- OWL-S no distingue entre coreografía y orquestación y la definición de procesos no se basa en un modelo formal, mientras que en WSMO se hace una clara distinción entre ambos conceptos y se permite la definición de diversos modos de interaccionar con cada servicio.

2.6.4 **WSMX**

El Web Service Modeling Execution Enviornment (Bussler et al., 2005) es un entorno de ejecución para el descubrimiento, selección, composición, mediación, invocación y monitorización dinámica de servicios web semánticos. Se trata de una implementación de referencia basada en el modelo conceptual descrito en WSMO y que sirve como arquitectura para evaluar la idoneidad de los elementos que este modelo conceptual incorpora. De forma muy general, la funcionalidad de WSMX se resume en el poder de satisfacer el objetivo de un usuario a través de la selección dinámica de un servicio web apropiado, la mediación de los datos que necesitan ser comunicados a este servicio y su invocación. En la Figura 20, se muestran los distintos componentes que constituyen WSMX.

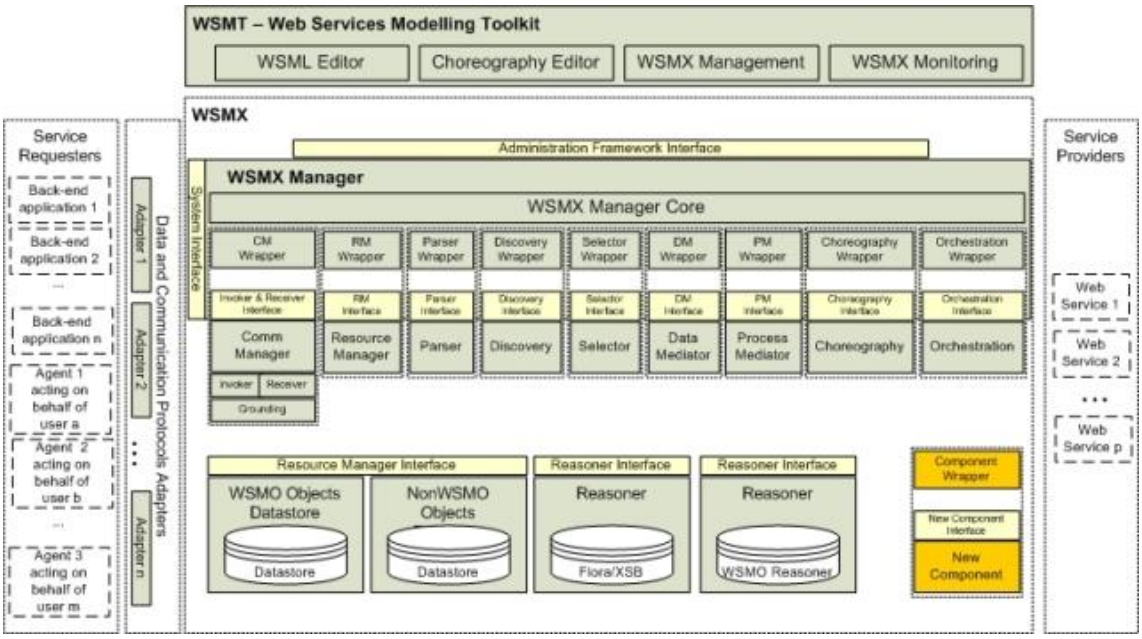


Figura 20. Arquitectura de WSMX

La funcionalidad global del sistema se encuentra dispersa entre los componentes que lo constituyen:

- El ''WSMX Manager Core'' se encarga de coordinar las actividades de los restantes componentes del sistema así como las interacciones que se producen entre ellos. Los datos que se manejan dentro de WSMX se representan mediante un formato interno que toma la forma de un evento con un tipo y un estado. Dicho componente, núcleo de la arquitectura de WSMX, gestiona desde un punto de vista lógico el procesamiento de todos los eventos, trasladándolos a los diferentes componentes de WSMX.
- ''Discovery'' es el componente encargado de proporcionar la funcionalidad necesaria para hacer corresponder la descripción de los servicios web semánticos con los objetivos indicados por los usuarios.
- El componente ''Selection'' se utiliza para encontrar los servicios más apropiados cuando se dé el caso de que se hayan encontrado numerosos servicios que satisfacen el objetivo previamente establecido por el usuario.
- ''Data Mediator'' proporciona los medios para transformar datos basados en conceptos de una ontología en datos basados en conceptos de otra ontología. El proceso de mapeo se basa en reglas definidas entre los conceptos de las ontologías fuente y destino. Si se da el caso en el que se precisa mediación de datos y estos datos no se encuentran en formato XML, se hace uso del conversor XML para traducir los resultados del mediador en XML. Esto es necesario porque las invocaciones de servicios web se realizan vía SOAP y el formato de los mensajes para SOAP es XML.
- Para ayudar en la resolución de los problemas de heterogeneidad que pueden aparecer durante la invocación de servicios web basados en coreografía existe el componente ''Process Mediator''. Este componente se encarga de asegurar que los procesos públicos del servicio invocador y del invocado se corresponden a la perfección.
- El componente ''Parser'' está basado en un compilador y un parseador de mensajes. El compilador parsea mensajes WSML recibidos desde el WSMO Editor en la capa de interfaz con el usuario. Tras esto valida estos mensajes contra WSMO y, entonces, almacena los elementos del mensaje en el repositorio de WSML. Los elementos compilados por WSMX son los metadatos de servicios web, ontologías y mediadores. Una vez que todos estos elementos han sido compilados a WSMX, están disponibles para su uso durante la ejecución de los objetivos enviados a WSMX. Por otro lado, el parseador de mensajes parsea los mensajes WSML que contienen los objetivos enviados a WSMX. Cada objetivo es parseado y almacenado de forma persistente. La diferencia entre el parseador de mensajes y el compilador es que, mientras este último maneja los meta-datos de servicios Web, ontologías y mediadores, el parseador funciona sobre instancias de objetivos.
- Los ''Adapters'' permiten a aplicaciones que no pueden comunicarse directamente con las interfaces provistas por WSMX establecer una comunicación con el sistema.
- El ''Choreography Engine'' se encarga de mediar entre los patrones de comunicación del solicitante y los del proveedor del servicio.
- El ''Reasoner'' proporciona servicios de razonamiento que permiten realizar los procesos de descubrimiento, mediación, validación de posibles composiciones de

servicios o determinación de si un servicio compuesto en un proceso es ejecutable en un contexto dado.

- ''Communication Manager'' es el componente que hace posible que WSMX se comunique con servicios web externos disponibles en la red, enviando los mensajes de solicitud necesarios y recibiendo la repuesta. Este componente se compone a su vez de tres elementos:
  1. ''Grounding''. Permite la trasformación entre los datos semánticos disponibles en WSMX y los datos en formato XML o RDF utilizados en las comunicaciones externa y viceversa.
  2. ''Invoker''. Utilizado cuando se precisa ejecutar un Servicio Web externo.
  3. ''Receiver''. Como su propio nombre indica, recibe la respuesta y la devuelve al formato semántico interno de WSMX.
- Por último, el ''Resource Manager'' se trata del componente responsable de gestionar los repositorios donde se almacenan las definiciones de objetivos, servicios web, ontologías y mediadores en WSMX.

2.7 Interoperabilidad

A lo largo de todo el documento se han proporcionado diferentes definiciones de interoperabilidad, qué problemas conlleva, que beneficios entraña la resolución de los mismos y el enfoque de la misma que se persigue con la investigación presentada. En este bloque del estado del arte se proporciona información de conceptos, lenguajes, iniciativas y trabajos desarrollados que giran en torno a la interoperabilidad. Todos ellos presentan diferentes enfoques y proporcionan soluciones o herramientas que pueden resultar de utilidad a la hora de enfrentarse a los retos que se proponen.

2.7.1 Interoperabilidad en Cloud Computing

El avance notable de tecnologías como la computación distribuida, Internet, grid computing, HPC, data centers, han posibilitado que Cloud Computing forme parte de un nuevo modelo de computación y de negocios. No obstante la convergencia de todas ellas funcionando en una gran infraestructura de tecnologías de la información, obliga a los especialistas a resolver muchísimos problemas de interoperabilidad. Cloud Computing es vista por muchos desarrolladores como la Cuarta Generación de aplicaciones y se espera que sea la forma en que se construirán las mismas durante los próximos años. Es un modelo emergente y de carácter empírico y los desarrollos en muchos casos se realizan en la capa SaaS, utilizando las APIs o servicios de las empresas proveedoras como pueden ser Amazon o Salesforce, por nombrar algunas.

Presenta muchas ventajas al usuario final como gran flexibilidad, ROI muy conveniente y TCO reducido. Debido a que Cloud no tiene estándares definidos, sus estilos son diferentes y, por tanto, las soluciones también lo son. Alcanzar una arquitectura interoperable que funcione adecuadamente para plataformas heterogéneas no resulta una tarea fácil, el objetivo que se

pretende alcanzar con esta línea de investigación es la construcción de soluciones que permitan integrar aplicaciones para construir Clouds Públicas y Abiertas.

Los modelos de programación para Cloud no están estandarizados, no existe una plataforma más importante que otra (al menos por ahora) y los tipos de aplicaciones que contienen también pueden ser muy variadas. En algunos casos de cloud ad-hoc se utilizan soluciones de grid o de SOA, en otros casos los algoritmos y extensiones de MapReduce, o también aplicaciones HPC tradicionales. Cuando se trabaja sobre cloud de proveedores como Amazon, Google, Abiquo, Universidad de California (Eucaliptus), IBM o Microsoft (Windows Azure), Union Europea (OpenNebula), Universidad de Chicago (Ninbus), Manjrasoft (Aneka), los desarrollos son programados de acuerdo a las APIs provistas o a los lenguajes soportados. Todo esto indica que por un tiempo convivirán distintas formas de SaaS.

En el ámbito del Cloud Computing, existen tres términos íntimamente ligados: portabilidad, interoperabilidad y seguridad. Esto se debe a que evitar la dependencia absoluta de un solo proveedor implica algo más que tener acceso a precios competitivos o a un mejor servicio. Contar con un solo proveedor supone un riesgo, especialmente en lo que concierne a la disponibilidad de servicios y datos.

La necesidad de portabilidad e interoperabilidad se ha solucionado mediante la estandarización, lo cual permite la interoperabilidad mediante abstracción (o intermediación) y la portabilidad mediante la conversión en un entorno con muchos estándares. Esta necesidad de interoperabilidad está presente en todas las capas del modelo, sin embargo en donde posee mejores soluciones es el IaaS mediante el uso de ambientes de virtualización.

El reto más importante a la interoperabilidad es la incompatibilidad actual de las APIs de gestión para cargar, descargar, inspeccionar, configurar y ejecutar acciones (p. ej. crear e iniciar nuevas instancias). Cada proveedor tiene su propia API para evitar que el software de orquestación funcione con distintos proveedores de servicios. Existen varias soluciones para este problema, como el Open Grid Forum que ha definido el estándar OCCI (Open Cloud Computing Interface); Eucalyptus, que emula la interfaz de los servicios web de Amazon como estándar válido; y VMware que ha desarrollado la API vCloud, la cual ofrece una base de interoperabilidad entre los proveedores de servicios basados en VMware (y posiblemente otros proveedores en el futuro).

La mayoría de los proveedores renuncian a la estandarización oficial porque quieren (y necesitan) moverse rápidamente en este mercado en constante evolución. Sin embargo, el hecho de que no se adopte una API única para todo el sector no tiene por qué impedir la portabilidad e interoperabilidad.

En cuanto al concepto de plataforma como servicio (PaaS), la portabilidad e interoperabilidad constituyen un desafío aún mayor. Los formatos de los datos para los servicios de plataforma suelen ser completamente diferentes. Así por ejemplo, Windows Azure suministra servicios de bases de datos y contenedores de aplicaciones .NET. Las aplicaciones y los datos de Azure no son compatibles con Google AppEngine y viceversa. La única forma de evitar la dependencia de un único proveedor cuando se utiliza PaaS es elegir una estructura facilitada por varios proveedores y evitar extensiones específicas de un proveedor (como las de Python de AppEngine). Una posible solución a este problema es el uso de una estrategia de abstracción parecida, donde se puedan desarrollar aplicaciones ejecutables en muchas soluciones PaaS.

El SaaS, supone el reto principal debido a la diversidad de datos y la imposibilidad de migración transparente entre proveedores. En este entorno, la conversión es una vía más práctica para la portabilidad que la estandarización.

En este ámbito, algunas teorías indican que surgirán varios estándares, de los cuales van a sobrevivir solo aquellos lo suficientemente robustos como para poder ser aplicados en múltiples ambientes, sin embargo, sigue siendo una tarea de gran complejidad y a largo plazo el desarrollo basado en una única opción lo suficientemente potente, sencilla y adaptable como para que sea adoptado por la gran mayoría de infraestructuras de computación en nube que existen actualmente y las que aparecerán en el futuro.

2.7.2 **Marcos de Interoperabilidad**

Interoperabilidad es la capacidad de los sistemas de tecnología de comunicación e información, y de los procesos de negocio para intercambiar datos y ser capaces de compartir información y conocimientos.

Un marco de interoperabilidad puede ser definido como un conjunto de normas y directrices que describen la forma en que las organizaciones han puesto de acuerdo, o deberían estar de acuerdo, para interactuar unos con otros. Un marco de interoperabilidad, por lo tanto, no es un documento estático y debe poder adaptarse con el tiempo a medida que las tecnologías, los estándares y los requisitos administrativos cambian.

2.7.2.1 ISO 15745 Framework for Application Integration

Describe un Application Integration Framework (AIF). Se trata de un conjunto de elementos y reglas para describir perfiles, que permiten un entorno común para la integración de aplicaciones y proveer el desarrollo de modelos para Application Interoperability Profiles (AIPs), y sus componentes (procesos, recursos e intercambio de información).

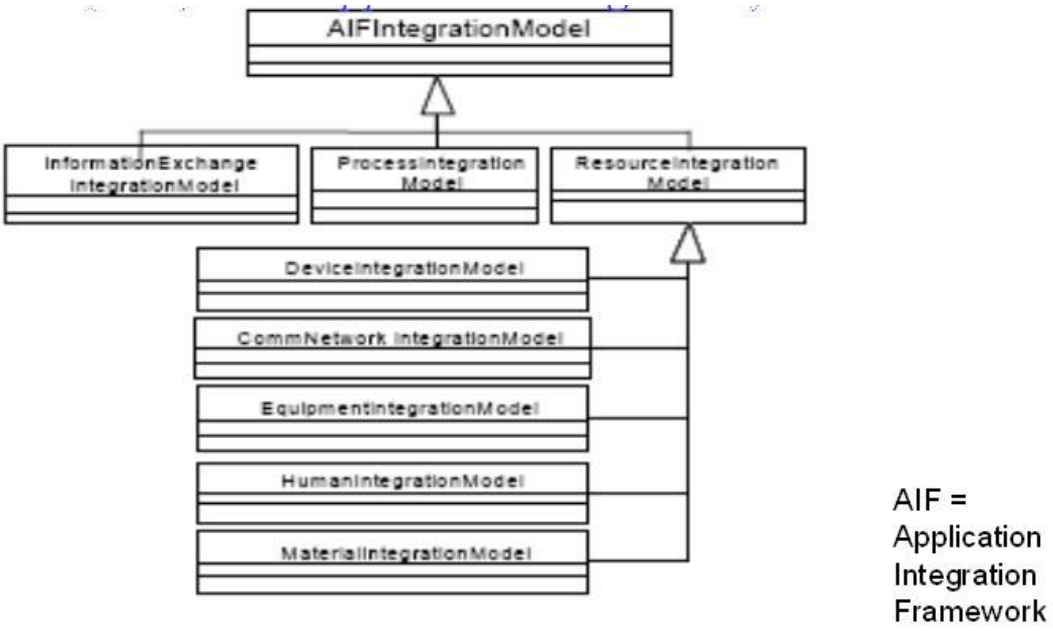


Figura 21. Modelo de Integración AIF

- **Número del Estándar:** ISO 15745-1-2003
  Título: Sistemas de automatización industrial e integración – la aplicación de sistemas abiertos marco de integración – Parte 1: Descripción de la referencia genérica
  Reemplaza el Estándar: ISO / FDIS 15745-1-2002
  Descripción: ISO 15745 define un marco de integración de aplicaciones (un conjunto de elementos y reglas para la descripción de los modelos de integración e interoperabilidad de aplicaciones. Define los elementos genéricos y reglas para describir los modelos de integración de aplicaciones y perfiles de interoperabilidad, junto con los perfiles de sus componentes), perfiles, perfiles de procesos de intercambio de información y de recursos.

- **Número del Estándar**: ISO 15745-2-2003
  Título: Sistemas de automatización industrial e integración – la aplicación de sistemas abiertos marco de integración – Parte 2: descripción de referencia para los sistemas de control basados en ISO 11898
  Descripción: ISO 15745-2:2003 define los elementos de tecnología y normas específicas para la descripción de los dos perfiles de comunicación de red y los aspectos relacionados con la comunicación de perfiles de dispositivos específicos para los sistemas de control basados en ISO 11898. En particular, ISO 15745-2:2003 describe plantillas específicas de tecnología de perfil para el perfil del dispositivo y la red de comunicación.

- **Número del Estándar**: ISO 15745-3-2003
  Título: Sistemas de automatización industrial e integración – la aplicación de sistemas abiertos marco de integración – Parte 3: descripción de referencia IEC 61158 para sistemas de control basados
  Descripción: ISO 15745-3:2003 define los elementos de tecnología y normas específicas para la descripción de los dos perfiles de comunicación de red y los aspectos relacionados con la comunicación de perfiles de dispositivos específicos según IEC 61158 basados en sistemas de control. En particular, ISO 15745-3:2003 describe plantillas específicas de tecnología de perfil para el perfil del dispositivo y el perfil de comunicación de red.

- **Número del Estándar**: ISO 15745-4-2003
  Título: Sistemas de automatización industrial e integración – la aplicación de sistemas abiertos marco de integración – Parte 4: descripción de referencia para los sistemas de control basados en Ethernet
  Descripción: ISO 15745-4:2003 define los elementos de tecnología y normas específicas para la descripción de los dos perfiles de comunicación de red y los aspectos relacionados con la comunicación de perfiles de dispositivos específicos para los sistemas de control basados en Ethernet. En particular, ISO 15745-4:2003 describe plantillas específicas de tecnología de perfil para el perfil del dispositivo y la red de comunicación 15745-4:2003

- **Número del Estándar**: ISO 15745-5-2007
  Título: Sistemas de automatización industrial e integración – la aplicación de sistemas abiertos marco de integración – Parte 5: descripción de referencia para los sistemas de control basados en HDLC
  Reemplaza el Estándar: ISO / DIS 15745-5-2005
  Descripción: ISO 15745-5:2006 define los elementos de la tecnología y normas específicas para la descripción de los dos perfiles de comunicación de red y de los aspectos relacionados con la comunicación de perfiles de dispositivos específicos para systems.iso control de HDLC basado 15745-5:2006.

2.7.2.2 CEN/ISO 11354 Requirements for establishing manufacturing enterprise process interoperability

La norma se basa principalmente en las aportaciones de varios proyectos europeos de desarrollo, llevados a cabo en el dominio de interoperabilidad empresarial. Inicialmente la red temática IDEAS (Interoperability Development of Enterprise Applications and Software) (Cunningham, Cunningham, & Fatelnig, 2003)se puso en marcha con el objetivo de elaborar una hoja de ruta para la interoperabilidad. Luego, dos importantes iniciativas relacionadas con el desarrollo de la interoperabilidad, ATHENA (Advanced Technologies for Interoperatbility of Heterogeneous Enterprise Networks and their Applications) (Ruggaber, 2006) e INTEROP (Interoperability Research for Network Enterprises Applications and Software) (Hervé Panetto, Scannapieco, & Zelm, 2004) se implicaron.

Despues, y basadas en ATHENA IP y INTEROP NoE, se crearon dos organizaciones para continuar con el desarrollo de la interoperabilidad empresarial: VLab (Virtual Laboratory) de INTEROP NoE, y EIC (European Interoperability Centre) creado por ATHENA IP.

Las necesidades de interoperabilidad empresarial se refieren a la capacidad de las empresas (o parte de ellas) para interactuar mediante el intercambio de información y otras entidades, como los objetos materiales, energía, etc. La interoperabilidad es un apoyo necesario para permitir que la colaboración empresarial suceda, y puede aplicarse tanto a las necesidades inter-e intra-empresa e incluye el concepto de empresa extendida, empresa virtual y sub-sistemas de una empresa. La interoperabilidad se considera como un concepto genérico, y por lo tanto se supone que los problemas comunes de interoperabilidad y las soluciones para superarlos, pueden ser identificados y desarrollados para cualquier empresa en particular.

El estándar CEN / ISO 11354 define un marco para la interoperabilidad empresarial y especifica los procesos y los metadatos que sustenta. Estos datos tienen que estar en su lugar para establecer o para permitir las soluciones de interoperabilidad de la empresa para la fabricación de Manufacturing Enterprise Processes (MEP) y sus modelos. El marco establece una base para la interoperabilidad en entornos unificados, integrados y federados, llamado enfoque interoperacional.

Los aspectos de interoperabilidad son cuatro: datos, servicio, proceso, y negocios. Los datos son utilizados por los servicios, incluidos los servicios web. Los servicios son empleados por los procesos para realizar negocios empresariales. Desde otro punto de vista, el objetivo de una empresa es ejecutar su negocio. Para realizar el negocio, se necesitan procesos. Los procesos emplean servicios, que a su vez necesitan de datos para realizar las actividades. Este contexto se ilustra en la Figura 22:

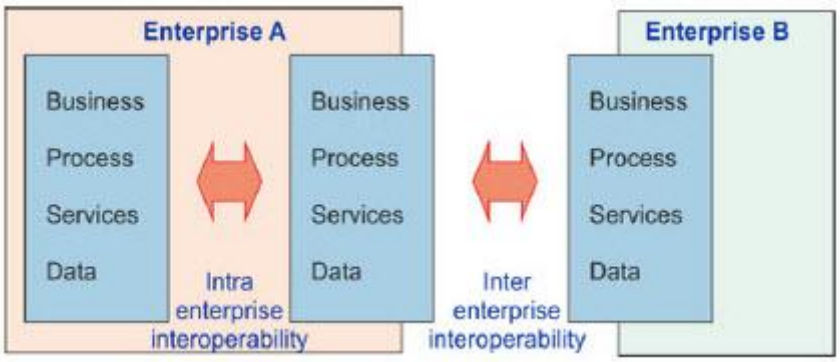


Figura 22. Categorías de interoperabilidad ATHENA 2007

Por último, se deben identificar las posibles incompatibilidades y desajustes que obstruyen la participación y el intercambio de información y otras entidades. Se definen tres categorías de barreras u obstáculos: conceptual, tecnológico y organizativo.

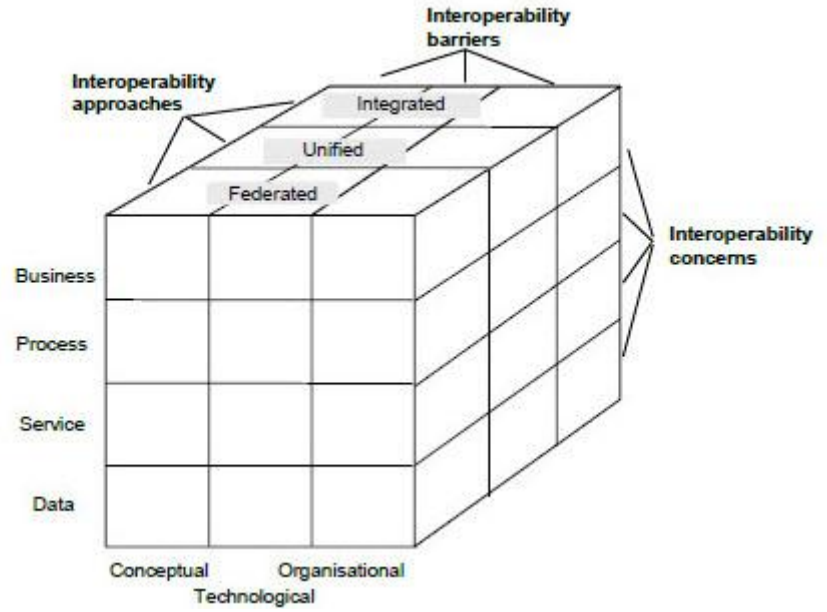


Figura 23. Marco de la Interoperabilidad Empresarial

El marco describe las diferentes categorías de interoperabilidad y sus necesidades particulares. Recoge el funcionamiento operacional de los MEPs, incluyendo la interoperabilidad de sus aplicaciones de soporte. Se centra en facilitar la comunicación en lugar de definir la propia comunicación y es independiente de tecnologías específicas.

El estándar se origina en los proyectos europeos ATHENA e INTEROP. La primera parte (Framework for Enterprise Interoperability) (Chen & Daclin, 2010) ha sido apoyado, al menos en parte, por el proyecto INTEROP. La segunda parte (Maturity Model for Enterprise Interoperability) (Guédria, Chen, & Naudet, 2009) se ha iniciado como un nuevo elemento de trabajo después de la finalización del proyecto. El trabajo se lleva a cabo por el CEN TC310 WG1 y por la ISO TC 184 SC 5 WG1.

2.7.2.3 ATHENA FP6 IP BIF: Business Interoperability Framework

La interoperabilidad empresarial se caracteriza por las relaciones de negocio de una empresa y sus socios externos, clientes, proveedores y prestadores de servicios. El objetivo del Business Interoperability Framework es describir los principales componentes de la interoperabilidad empresarial y esbozar cómo una empresa puede evaluar y mejorar su interoperabilidad empresarial.Para este propósito, el BIF distingue cuatro categorías (ver

Tabla 4):

- Gestión de las Relaciones Exteriores.
- Empleados y Cultura.
- Procesos de negocio colaborativos.
- Sistemas de Información.

Business Interoperability (Diseño organizativo de las relaciones comerciales)		
Categoria	**Perpesctiva**	**Descripcion**
Gestión de las Relaciones Exteriores	“¿Como administramos y controlamos las relaciones comerciales?” (Governance Perspective)	Organizaciones interoperables gestionan y monitorizan sus relaciones comerciales.
Empleados y Cultura	"¿Cómo nos comportamos con nuestros socios?" (Behavioural Perspective)	Organizaciones interoperables promueven las relaciones con socios comerciales a nivel individual, colectivo y organizativo.
Procesos de Negocios Colaborativos	"¿Cómo podemos colaborar con nuestros socios?" (Operational Perpesctive)	Organizaciones interoperables pueden establecer y gestionar de forma rápida y barata las colaboraciones electrónicas con sus partners
Sistemas de Informacion	“¿Cómo puedo conectar con mis socios?” (Technical Perspective)	Sistemas ICT (Information and Communications Technology) pueden vincularse unos con otros de forma rápida y barata, y atender asi la estrategia de cooperación de la organización.
Contingencias (Factores que impactan en el diseño organizativo)		
Categoria	**Perpesctiva**	**Descripcion**
Contingencias Internas	“¿Cuáles son las características de una relación comercial?”	Los objetivos de cooperación y las características transaccionales impactan en el nivel optimo de interoperabilidad comercial
Contingencias Externas	“¿Qué factores ambientales afectan las relaciones comerciales?”	La madurez del E-Business, la legislación y la dinámica de la industria determinan las precondiciones en este contexto.

Tabla 4. Business Interoperability Framework – Categorias y Contingencias

Construido sobre la Contingency Theroy of Organizations (Donaldson, 2001), BIF postula que el diseño inter-organizacional óptimo se ajusta a contingencias externas (ambiental) e internas.

2.7.2.3.1 NIVELES DE LA INTEROPERABILIDAD COMERCIAL EN BIF

La idea de interoperabilidad no se ajusta a opciones binarias como "sí" o "no", si no que tiene múltiples facetas. Por consiguiente, existe una necesidad de distinguir diferentes niveles de interoperabilidad. La estructura del Business Interoperability Framework se inspira en los modelos de excelencia existentes, como el Modelo EFQM Excellence Model (Maderuelo Fernández, 2002) o el Capability Maturity Model (Paulk, 2002).

Sin embargo, a diferencia de EFQM y CMM, un mayor nivel de interoperabilidad de los negocios no es necesariamente un signo de la excelencia o madurez, debido al hecho de que el nivel óptimo de la interoperabilidad depende del ajuste entre la interoperabilidad y sus contingencias. El mayor nivel de interoperabilidad comercial representa el valor máximo, es decir, el hecho de que una empresa sea totalmente interoperable en el sentido de que las nuevas relaciones comerciales pueden establecerse con un coste mínimo. Este hecho podría no representar necesariamente el nivel óptimo para una organización en concreto, ya que podría ser el resultado de un exceso de inversión en interoperabilidad. Para reflejar la relación óptima, se propone el llamado "neutral" de los niveles del Business Interoperability.

No.	Business Interoperability	Descripcion
(1)	**Ninguna**	Sin conocimiento de relaciones externas; Interacción con socios externos no planeada o realizada ad-hoc
(2)	**Minima**	Sin previsiones de Interoperabilidad; Diseño individual para cada relación externa
(3)	**Moderada**	Se entiende la relevancia de la Interoperabilidad comercial; Se han tomado medidas para mejorar la interoperabilidad, pero sigue habiendo un gran margen de mejora
(4)	**Competente**	Las relaciones externas se diseñan para mejorar la Interoperabilidad comercial; Solo carece de algunos factores en el camino a la total interoperabilidad
(5)	**Totalmente Interoperable**	Máximo nivel de interoperabilidad comercial; Las relaciones externas pueden establecerse a ningún o mínimo coste.

Tabla 5. Los cinco niveles de la Interoperabilidad Comercial en BIF

2.7.2.4 CEN-ISSS EBIF CEN eBusiness Interoperability Roadmap

Como apoyo a la plataforma europea para la consideración de soluciones de interoperabilidad relacionadas con el comercio electrónico, y recomendar estrategias sobre las actividades de normalización necesarias para lograrlo, CEN lanzó a principios de 2005, y siguiendo las recomendaciones del eBusiness Focus Group, el eBusiness Interoperability Forum (EBIF). Entre los participantes, los proveedores de TI, los usuarios finales de la industria, representantes de las PYME, las administraciones públicas, la Comisión Europea y representantes de los consorcios de estándares.

Las actividades de EBIF se fusionan en las actividades del eBusiness Coordination Group.

- UN/CEFACT and UBL convergence - May 2012
- eBusiness Roadmap (2006-2008)
- ebXML for managers (2005)
- CWA 16093 - Feasibility Study for a Global eBusiness Interoperability Test Bed (GITB)

Cinco aspectos estratégicos están identificadas en esta hoja de ruta:

- **eBussines frameworks y estándares**: garantizar el marco de interoperabilidad transfronteriza en el comercio electrónico.
- **eContent interoperability**: define reglas para el mantenimiento y la interoperabilidad del eContent (contenidos digitales).
- **Soluciones inclusivas**: verifiquen que los marcos y las normas de comercio electrónico se encuentran específicamente los requisitos europeos en todos los niveles.
- **Frameworks de confianza**: que los usuarios puedan confiar en sus socios comerciales y sus sistemas de comercio electrónico.
- Dar apoyo a las implementaciones de interoperabilidad de referencia, soluciones sólidas, asequibles y que permitan el uso de interfaces abiertas, las cuales puedan ser reutilizadas por una amplia comunidad.

Además de los aspectos estratégicos arriba mencionados, se necesitan medidas horizontales para facilitar y apoyar su implementación. El uso de software de código abierto se fomentará en reuniones los objetivos clave. Tanto el software de código abierto y cerrado, deben ser capaces de utilizar igualmente los componentes e interfaces de interoperabilidad.

2.7.2.5 UN / CEFACT eBusiness Interoperability Framework

En el marco del Consejo Económico y Social de las Naciones Unidas, la United Nations Economic Commission for Europe (UNECE) sirve como punto focal para las recomendaciones sobre facilitación y las normas del comercio electrónico, que abarca los procesos de negocios comerciales y gubernamentales que pueden fomentar el crecimiento en el comercio internacional y servicios relacionados. En este contexto, se estableció como una filial el United Nations Centre for Trade Facilitation and Electronic Business (UN / CEFACT), órgano

intergubernamental del Comité del UNECE sobre el comercio, con la misión de elaborar un programa de trabajo de relevancia mundial para lograr una mejor coordinación en todo el mundo y la cooperación en estas áreas.

UN / CEFACT apoya las actividades dedicadas a mejorar la capacidad de las empresas, organizaciones comerciales y administrativos, de economías desarrolladas o en vías de desarrollo, para el intercambio de productos y servicios relevantes de forma eficaz. Su enfoque principal es facilitar las transacciones nacionales e internacionales, a través de la simplificación y homogenización de los procesos, procedimientos y flujos de información, y así contribuir al crecimiento del comercio mundial. Esto se logra mediante:

- Analizar y comprender los elementos clave de los procesos, procedimientos y operaciones internacionales, y trabajar por la eliminación de sus barreras.
- Desarrollando métodos para facilitar los procesos, procedimientos y operaciones, incluyendo el uso pertinente de las tecnologías de la información.
- Promover el uso de estos métodos y las buenas prácticas asociadas, a través de canales como asociaciones gubernamentales, industria y empresas de servicios.
- Coordinar el trabajo con otras organizaciones internacionales como la Organización Mundial del Comercio (WTO), la Organización Mundial de Aduanas (WCO), la Organización para la Cooperación y el Desarrollo Económico (OECD), la Comisión de las Naciones Unidas para el Derecho Mercantil Internacional (UNCITRAL) y la Conferencia de las Naciones Unidas sobre Comercio y Desarrollo (UNCTAD), sobre todo en el marco de un Memorando de Entendimiento para una Asociación Mundial para la Facilitación del Transporte y el Comercio.
- Asegurar la coherencia en la elaboración de normas y recomendaciones mediante la cooperación con otras partes interesadas, incluidas las organizaciones internacionales, intergubernamentales y no gubernamentales. En particular, para las Normas de la UN / CEFACT, esta coherencia se ve facilitada por la cooperación con la Internacional Organization for Standardization (ISO), la Comisión Electrotécnica Internacional (IEC) y la Unión Internacional de Telecomunicaciones (ITU) y las organizaciones no gubernamentales pertinentes (ONGs), especialmente en el contexto del memorando del Memorandum of Understanding (MoU) ISO / IEC / ITU / UNECE.

El documento original titulado "Mandate, Terms of Reference and Procedures for UN / CEFACT" (TRADE/R.650) fue aprobado por WP.4, el predecesor del Centro de las Naciones Unidas para la Facilitación del Comercio y el Comercio Electrónico (UN / CEFACT) en su última reunión en septiembre de 1996, y posteriormente fue aprobado por el Comité de Desarrollo del Comercio en diciembre de 1996. Posteriormente, los cambios organizativos, un cambio de nombre y de la experiencia adquirida en el funcionamiento del Centro organización dio lugar a modificaciones en el documento original, siendo la última revisión 4, la que fue aprobada por el Pleno del UN / CEFACT en junio de 2006.

2.7.2.6 OMG Service Driven Architecture

Se trata de poner en relieve la importancia y valor de aprovechar los principios y conceptos fundamentales detrás de las "Event Driven Architectures" (EDA) (Sr & Sr, 2005), en favor de las "Service Driven Architectures", más comúnmente conocido como "Service Oriented Architectures" (SOA) (Erl, 2005), los servicios basados en XML. Se quiere remarcar el hecho de que EDA aumenta a SOA y MDA "Model Driven Architecture", todos los enfoques se complementan y soportan soluciones agiles centradas en el usuario.

En Arquitecturas Orientadas a Servicios (SOA), la arquitectura de una empresa es diseñada y enfocada a Service Driven, donde cada servicio es autónomo y puede ser considerado como un Service Building Block (SBB). Usando una analogía entre el concepto de servicio y un proceso de negocio, en SOA, SBB de acoplamiento flexible son orquestados en procesos de negocios para conseguir las metas del cliente y/o de la organización.

Más que un componente de código reutilizable, una SBB se convierte en parte de un programa en ejecución que puede ser invocado por un cliente sin tener que incorporar el código en sí. Un SBB, por definición, es reutilizable y reemplazable, es decir, un servicio de SBB se vuelve a utilizar una y otra vez por los otros servicios por la funcionalidad que proporciona, y el servicio de SBB puede ser sustituido por otro (implementación de otro proveedor). En SOA, las SBB se pueden clasificar en servicios básicos / fundamentales, servicios de administración, servicios de seguridad, servicios de oficina, servicios de portal, etc. Cabe también señalar que la SBB ofrecen funcionalidades específicas para la empresa y proyectos, como por ejemplo, un Digital Rights Management Service (DRM) como SBB sólo se implementaría una vez en una arquitectura empresarial y se podría reutilizar en otros proyectos. En SOA, el flujo de comunicación está cerrado a nuevas entradas imprevistas una vez que el flujo de la comunicación ha comenzado, es decir, está bien definido y sus límites bien establecidos.

Event Driven Architecture, abarca los mecanismos de coordinación de los solicitantes y los proveedores de servicio, productores y consumidores de datos, "sensors" y "responders" de eventos software con nivel variable de dependencia de comunicación, con un espectro variable de correlación de mensajes y con opciones variables de calidad de servicio. EDA soporta flujos de mensajes dinámicos, paralelos y asíncronos, y por lo tanto reacciona a las entradas externas que son impredecibles. EDA puede coordinar de forma sincrónica o asincrónica entre software endpoints, y posiblemente proporcionar tanto acceso síncrono y asíncrono entre los participantes. En EDA, flujos simultáneos de ejecución, pueden ejecutarse de forma independiente para cumplir con una petición del cliente o tareas del sistema.

2.7.2.6.1 Proposición de Valor de Evenetos (EDA) para servicios SOA

Al establecer el escenario y ofrecer una escala de validación, los autores presentan primero cómo EDA se completa y complementa el espacio de soluciones de la meta-arquitectura con SOA y MDA.

En primer lugar, se cree que MDA, SOA y EDA forman el eje de la estrategia de arquitectura que compone la evolución de cualquier arquitectura de software en el espacio de la solución

arquitectónica. Esta creencia deriva del hecho de que los tres elementos fundamentales ortogonales de cualquier software son la estructura, la función y los datos. Los autores creen que MDA, SOA y EDA son conceptos ortogonales que se desprenden de estas tres formas fundamentales.

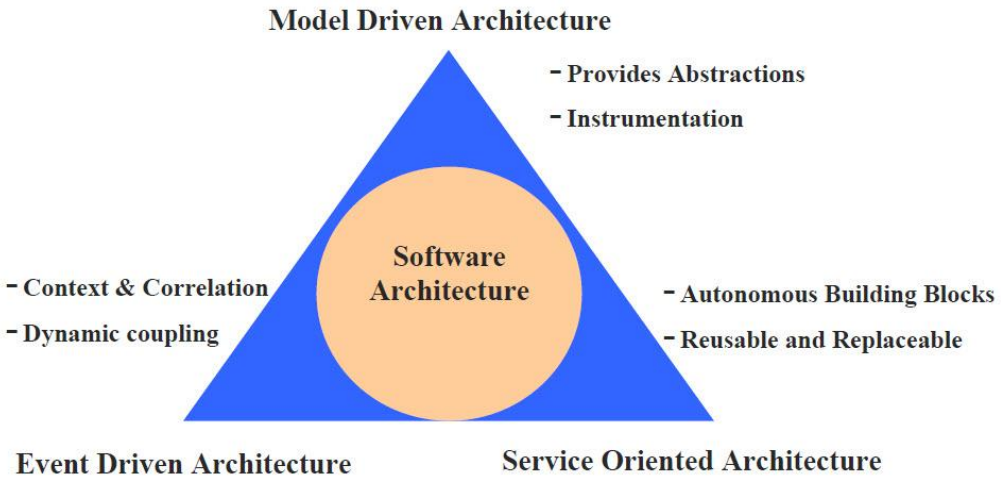


Figura 24. Axiomas de la estrategia arquitectónica

En segundo lugar, en la construcción de una escala de validación, se considera a Usuarios, Negocio y Sistema como las tres partes arquitectónicas primordiales. "User" representan las preocupaciones de dependencia externas al sistema, "Business Domain" representa las preocupaciones de funcionalidad internas que componen el sistema, y "Builder" representa los problemas de desarrollo que existe en la evolución del sistema.

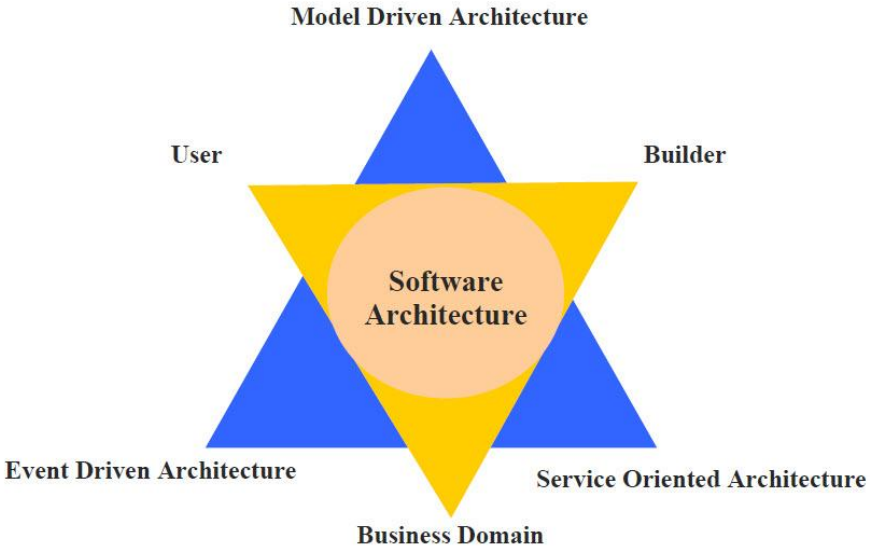


Figura 25. Perspectivas de Usuario, Diseño y Negocio

2.7.2.7 iDABC European Interoperability Framework for Pan-European eGovernment Services

El programa IDABC (*Interoperable Delivery of Pan-European eGovernment Services to Public Administrations, Business and Citizens* - prestación interoperable de servicios paneuropeos de administración electrónica al sector público, las empresas y los ciudadanos) tiene por objeto prestar servicios paneuropeos de administración electrónica a las administraciones públicas, las empresas y los ciudadanos. El objetivo consiste en mejorar la eficacia de las administraciones públicas europeas y la colaboración entre ellas.

IDABC es un programa de administración electrónica establecido para el período 2005-2009.

Sustituye al programa IDA (Interchange of Data between Administrations – intercambio de datos entre administraciones), con un campo de acción más amplio. Cubre los objetivos del programa IDA pero tiene por objeto también crear servicios paneuropeos de administración electrónica para las empresas y los ciudadanos.

IDABC entra en el marco de las iniciativas eEurope 2005 e i2010.La interoperabilidad y las normas abiertas siguen siendo campos de acción prioritarios, a los cuales se añaden los nuevos servicios paneuropeos que deben crearse.

2.7.2.7.1 Objetivos

El programa IDABC busca apoyar y promover la puesta a punto de servicios paneuropeos de administración electrónica, así como las correspondientes redes telemáticas interoperables.

El programa tiene también como objetivos:

- Permitir el intercambio de información entre las administraciones públicas, y entre éstas y las instituciones comunitarias.
- Facilitar el suministro de servicios paneuropeos a las empresas y a los ciudadanos, teniendo en cuenta sus necesidades.
- Lograr la interoperabilidad entre los distintos ámbitos de acción, en particular, basándose en un marco de interoperabilidad europeo.
- Promover la difusión de buenas prácticas y fomentar la elaboración de soluciones telemáticas innovadoras en las administraciones públicas.

El programa IDABC incluye proyectos de interés común que permiten la aplicación de la legislación comunitaria y la mejora de la cooperación interinstitucional.

El programa contiene también medidas horizontales que contemplan la instauración de servicios paneuropeos horizontales de administración electrónica y servicios de infraestructura, en particular, en favor de la interoperabilidad.

2.7.3 **Orquestación y Coreografía**

Los términos orquestación y coreografía describen dos aspectos de la creación de procesos de negocio para la composición de servicios web. Ambas definiciones se solapan en cierto modo ya que sus significados son a menudo intercambiados o difíciles de describir. Según (Peltz, 2003) se puede definir la orquestación como un proceso de negocio ejecutable que interactúa tanto con servicios web externos como internos. En este caso las interacciones ocurren a nivel de mensaje e incluyen la lógica de negocio y el orden de ejecución de las tareas. Por otro lado el mismo autor define la coreografía como la secuencia de mensajes entre diferentes partes y fuentes (normalmente mensajes públicos intercambiados entre los servicios web) en lugar de un proceso de negocio específico que ejecuta una única parte como ocurre en la orquestación. La perspectiva que se establece en la coreografía es más colaborativa y permite a cada parte involucrada describir sus partes en la interacción.

El W3C's Web Services Choreography Working Group define sin embargo la coreografía como la definición de las secuencias y condiciones bajo las cuales múltiples agentes cooperan de forma independiente intercambiando mensajes con el fin de realizar una tarea y alcanzar un su estado objetivo. La coreografía de los servicios web se preocupa de las interacciones entre los servicios y sus usuarios. Cualquier usuario de un servicio web, automatizado o de otra manera, es un cliente de dicho servicio. Esos usuarios puede, en ocasiones, ser otro servicio web, aplicaciones o humanos. La orquestación es definida en este caso como la secuencia y condiciones en las que un servicio web invoca otro servicio web para poder realizar una tarea útil, en otras palabras, la orquestación es el patrón de interacciones que un agente de servicio web debe seguir para alcanzar su objetivo.

Otro tercer grupo de autores prefiere sustituir las palabras orquestación y coreografía por las de conversación y coordinación por considerarlas más apropiadas. Aunque en el fondo las definiciones son prácticamente las mismas, a determinadas personas les resulta más sencillo visualizar los procesos utilizando esta terminología.

La Figura 26 muestra gráficamente la relación entre los servicios web tanto en orquestación como en coreografía:

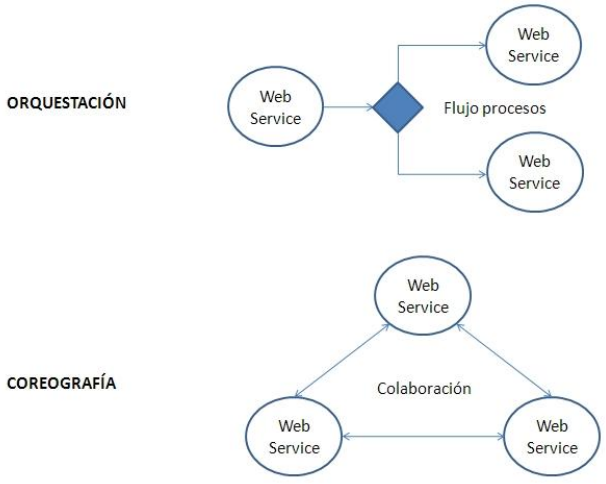


Figura 26. Orquestación vs. Coreografía

Una vez definidos estos dos importantes conceptos que permiten conocer de forma más amplia el alcance y potencial de los servicios web semánticos, es momento de comentar las dos tendencias más importantes que se dan en esta línea. Teniendo en cuenta el actual estado del arte se tienen que, por un lado, los esfuerzos de la industria tratan de establecer el terreno para hacer que las actividades comerciales se den sobre las bases de los servicios web y, por otro lado, la investigación académica se dedica a proveer a los Servicios Web de tecnología que permita las anotaciones para que así puedan soportar mejor la localización (discovery), composición y ejecución de los actuales servicios web. En paralelo a lo anterior, el W3C está trabajando para unificar esas dos posturas a través de distintos grupos de interés como el Web Services Addressing Working Group, el Web Services Choreography Working Group y el Semantic Web Interest Group/Best Practices and Deployment Working Group.

En cuanto a la composición y ejecución de servicios web, la orquestación representa la fusión de ambos elementos conceptuales, aportando un importante paso hacia una mayor automatización de los procesos de negocio. Sin embargo, la integración real de los procesos de negocio entre las empresas no puede basarse exclusivamente en mensajes y protocolos estándares. Las interacciones de negocio requieren la ejecución de procesos de larga duración y por ello se han definido un abrumador número de lenguajes para la composición de servicios web. Estos lenguajes desarrollados sobre WSDL soportan la definición de procesos de ejecución ya que conservan los estados sobre distintas ejecuciones de Servicios Web. Entre otras iniciativas cabe destacar BPEL4WS, BPML, ebXML, y WS-CDL. BPEL4WS (Andrews et al., 2003) es el resultado de una iniciativa conjunta entre BEA, IBM, Microsoft, SAP AG y Siebel Systems. Provee de un lenguaje para la especificación formal de procesos de negocio y protocolos de interacción entre negocios. Extiende, por tanto, el modelo de interacción de Servicios Web permitiendo que la definición de los procesos de ejecución recaiga exclusivamente en recursos de Servicios Web y en datos XML.

Otra especificación, ebXML (Electronic Business using eXtensible Markup Language), que empezó en 1999 como iniciativa de OASIS y de la agencia United Nations/ECE CEFACT. ebXML es una conjunto modular de especificaciones que trata de dar soporte a los comportamientos de negocio de las empresas a través de Internet. Parte del ebXML es BPSS, (Business Process Specification Schema) de 2001, un lenguaje similar al BPEL4WS o BPML. Finalmente, el W3C que trabaja en la estandarización de la coreografía de procesos de negocio de Servicios Web, propuso un borrador de dicho lenguaje denominado Web Services Choreography Description Language (Kavantzas et al., 2005).

(Aalst, 2003) presenta un significativo análisis de los lenguajes estándares más relevantes para la composición de servicios web. Este análisis está basado en un framework basado en flujos de trabajo recurrentes y en patrones de comunicación, que proveen un espacio de trabajo formal para testear y comparar la expresividad de cada uno de estos lenguajes. La principal conclusión de estos estudios es que sus capacidades son bastante similares lo que explica porque no existe todavía una decisión final en apoyar a uno de estos lenguajes como estándar para la industria.

2.7.3.1 WS-CDL

Web Service Choreography Description Language (WS-CDL) (Kavantzas et al., 2005) especifica un lenguaje declarativo que describe colaboraciones entre servicios web (peer-to-peer) mediante la definición del comportamiento externamente observable de cada participante de un proceso de negocio o transacción. Es una propuesta del Web Service Choreography Group perteneciente al W3C.

La utilización de WS-CDL promueve el común entendimiento entre los participantes en los servicios web, garantiza automáticamente la conformidad, asegura interoperabilidad e incrementa la robustez. Además también conlleva beneficios como reducir el coste de implementación de servicios web debido a la aseguración del comportamiento esperado, incrementa su utilidad, su construcción más robusta y habilita una interoperabilidad más efectiva de los servicios web a través de contratos entre varias partes relativos a su comportamiento.

Los comportamientos observables son definidos desde un punto de vista global, no desde el de un participante particular. Como resultado de esto, las reglas de colaboración pueden ser definidas y consensuadas por los participantes involucrados.

WS-CDL se utiliza para describir coreografías de servicios web. Con WSDL, que es parte de la capa de descripción, la funcionalidad de un participante puede ser descrita pero basada en un modelo cliente-servidor sin estado. Este enfoque no es apropiado para colaboraciones complejas, compuestas de varias partes de forma peer-to-peer. El lenguaje WS-CDL hace posible la descripción de ese tipo de colaboraciones entre servicios y además es independiente de lenguajes de procesos de negocio ejecutables y específicos (orquestación).

Como ejemplo se puede comentar lo siguiente. Un cliente envía un mensaje RFQ a un proveedor. El proveedor le contesta con un presupuesto. El cliente envía entonces un mensaje para solicitar una serie de bienes y el proveedor confirma la orden. En WS-CDL en primer lugar se debe definir la coreografía global OrdenarBienes. A continuación se identifican los roles que van a interactuar: Cliente y Proveedor. Tras esto se definen las relaciones entre ellos y la forma en la que se van a comunicar. Finalmente se definen los canales e interacciones entre ellos. Estos pasos deben escribirse en una sintaxis XML definida en el WS-CDL XML Schema.

Como se comentaba, WS-CDL se suele aplicar en entornos de proceso de negocio. Aquellos que involucran un número elevado de organizaciones diferentes o procesos independientes que colaboran mediante la utilización de servicios web pueden resultar muy provechosos si se encuentran integrados adecuadamente.

Para que esto sea así, se ha producido una definición ''global'' de las condiciones y restricciones bajo la que los mensajes deben ser intercambiados que describe desde un punto de vista general el comportamiento observable común y complementario de todos los Servicios Web que participan. Cada participante puede emplear una definición global para construir y probar soluciones que se adapten a ello.

La principal ventaja de un enfoque global es que separa el proceso seguido por un negocio individual o sistema dentro de un ''dominio de control'' de la definición de la secuencia en la que cada negocio o sistema intercambia información con el resto. Esto significa que, siempre

que la secuencia ''observable'' no cambie, las reglas y la lógica seguidas dentro del dominio de control pueden cambiar y lo harán.

En escenarios del mundo real, entidades corporativas no siempre están dispuestas a delegar el control de sus procesos de negocio en compañeros de integración. La coreografía ofrece significados en los que las reglas de participación dentro de la colaboración pueden ser claramente definidas y acordadas conjuntamente. Cada entidad debe entonces implementar su porción correspondiente de coreografía según se determine en la vista común.

La Figura 27 muestra un posible uso del lenguaje de coreografías:

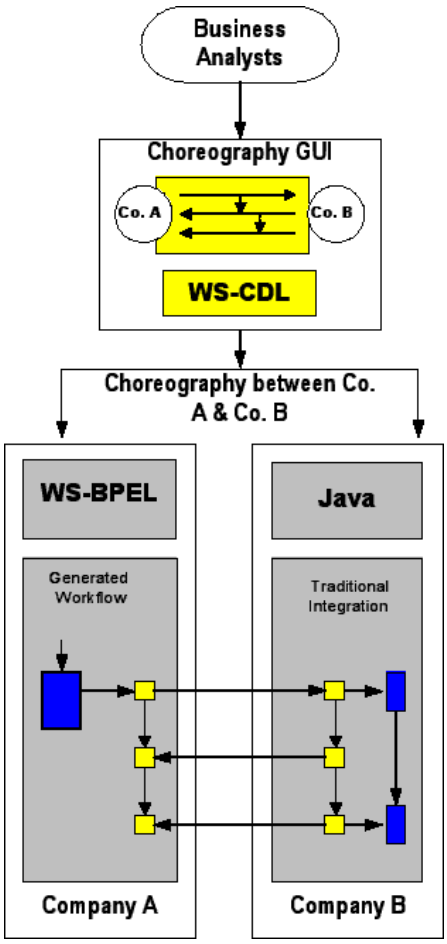


Figura 27. Integración de aplicaciones basadas en servicios web utilizando WS-CDL

2.7.3.1.1 Objetivos

El principal objetivo de la especificación de WS-CDL es definir un lenguaje declarativo, basado en XML, que defina desde un punto de vista global el comportamiento común, los intercambios de información que pueden ocurrir y las reglas de organización de acuerdos que han de satisfacerse.

De forma más específica los objetivos que se persiguen son los siguientes:

- **Reusabilidad**: La misma coreografía puede ser utilizada por diferentes participantes en diferentes contextos.
- **Cooperación**: Las coreografías definen la secuencia de intercambio de mensajes entre dos (o más) participantes independientes o procesos, describiendo cómo deben cooperar.
- **Colaboración entre múltiples participantes**
- **Semántica**: Las coreografías pueden incluir documentación legible para las personas.
- **Composición**: Las coreografías existentes podrán combinarse para formar otras nuevas que puedan ser reutilizadas en distintos contextos.
- **Colaboración dirigida por información**: Las coreografías describen cómo los participantes progresan en su colaboración a través de la grabación de la información intercambiada y los cambios en la información observable.
- **Sincronización de la información**: Las coreografías deben permitir a los participantes sincronizar la información observable.
- **Manejo de excepciones**: Las coreografías pueden definir cómo tratar condiciones excepcionales
- **Transaccionalidad**: El proceso o los participantes que toman parte en una coreografía puede trabajar de manera ''transaccional'' con la habilidad para coordinar el resultado de colaboraciones de larga vida, que incluyen múltiples participantes, cada uno con sus propias reglas de negocio y objetivos no observables.
- **Compatibilidad de la especificación**: Se intenta que esta especificación complemente o pueda trabajar con otras especificaciones del W3C.

2.7.3.1.2 Participantes

La especificación WSDL describe la funcionalidad de un servicio proporcionado por un participante basado en un modelo cliente-servidor conectado y sin estado. Las aplicaciones emergentes basadas en web requieren la capacidad para intercambiar mensajes en entornos peer-to-peer. En este tipo de entornos un participante representa a un cliente de servicios que proporciona otro participante, que es a su vez proveedor de los servicios requeridos por otros participantes, creando un servicio de mutua dependencia.

Un documento WS-CDL describe como un participante de un servicio web es capaz de abordar colaboraciones peer-to-peer con el mismo o diferentes participantes. En coreografía, la información siempre es intercambiada entre participantes.

Los roles, relaciones y canales definen la unión de los participantes en los procesos colaborativos de los servicios web.

Roles

Un rol enumera los comportamientos observables exhibidos por un participante para poder colaborar con otros participantes. Por ejemplo, el rol del comprador es asociado con la adquisición de bienes y servicios, y el rol de proveedor con la provisión de dichos bienes.

Participants

Un participante identifica un conjunto de roles relacionados. Como ejemplo se tiene una organización comercial que puede apropiarse tanto del rol de comprador como de vendedor.

Relationships

Una relación es la asociación de dos roles para la consecución de un objetivo. Una relación representa las posibles formas en las que dos roles pueden interactuar. Un ejemplo claro entre el comprador y el vendedor puede incluir:

- Una relación de compra, para la primera solicitud de adquisición de bienes.
- Una gestión de clientes, para permitir al proveedor proporcionar servicio y soporte después de que los bienes hayan sido vendidos.

Channels

Un canal descubre un punto de colaboración entre participantes mediante la especificación de dónde y cómo intercambiar la información. De forma adicional, la información del canal puede pasar entre participantes. Esto permite modelar cómo se determina el destino de los mensajes, estática y dinámicamente, cuando se colabora dentro de la coreografía.

Como ejemplo, un comprador puede especificar el canal de información a utilizar para enviar la información de entrega. El comprador pude entonces enviar el canal de información al vendedor, que lo reenvía al dependiente. El dependiente puede en ese momento enviar la información de entrega directamente al comprador empleando el canal de información originalmente proporcionado por el comprador.

Un canal tiene que describir obligatoriamente el rol y el tipo de una referencia a servicio web de un participante, siendo el objetivo de una interacción, que será usado entonces para determinar dónde y cómo enviar/recibir la información a/para del participante.

2.7.3.2 WSCI

Web Service Choreography Interface (WSCI) (Intalio et al., 2002) es un lenguaje de descripción de interfaces basado en XML, que describe no el flujo de tareas ejecutable, sino el flujo de mensajes intercambiados por un servicio web que participa de una coreografía de servicios. Por todo esto se dice que WSCI describe cómo las operaciones de los servicios web pueden ser coreografiadas en el contexto de intercambio de mensajes en el que participa el servicio web. WSCI trabaja en conjunto con WSDL y también con cualquier lenguaje de definición de servicios que exhiba las características de WSDL. Es una iniciativa de Sun Microsystems, BEA, Intalio, y SAP.

WSCI describe el comportamiento observable de un servicio web, expresado en términos de dependencias lógicas y temporales entre los mensajes intercambiados, ofreciendo reglas de secuencias, correlación, manejo de excepciones y transacciones sin la existencia de un servicio web ''director''. WSCI también describe el intercambio colectivo de mensajes entre los servicios web que interactúan, brindando así una vista global de las interacciones orientadas a mensajes. Se trata de un lenguaje declarativo que sólo permite al diseñador de sistemas comprender las interacciones de los servicios involucrados en una conversación. En ningún caso aborda temas relacionados con la implementación de los procesos que definen el comportamiento interno del servicio.

Se trata del primer paso que permite el mapeado de servicios como componentes que realizan el intercambio de mensajes o procesos. También se encarga de describir como la coreografía de esas operaciones pueden exponer información relevante como por ejemplo correlación de mensajes, excepciones en el manejo, descripción de transacciones y capacidades de participación dinámicas. Sin embargo, WSCI no asume que los servicios web provienen de diferentes compañías, como en business-to-business. Puede ser utilizado de la misma forma para describir interfaces de componentes que representan unidades organizacionales internas u otra aplicación de la empresa.

WSCI permite explicar las interdependencias entre las operaciones de los servicios web de forma que cualquier cliente:

- Pueda entender cómo interactuar con un determinado servicio en el contexto de un proceso dado
- Poder anticipar el comportamiento esperado de un servicio en cualquier punto del ciclo de vida del proceso

Siendo capaz de describir la interfaz dinámica de un servicio en el contexto de un proceso particular permite al desarrollador/arquitecto abstraerse desde la implementación y centrarse en el rol que desempeña un servicio web en el proceso.

Para formar parte de una colaboración útil y manejable entre servicios web, se deben permitir operaciones individuales hacer llegar suficiente información acerca de cómo pueden ser utilizadas en un escenario concreto de forma que se les permita participar en procesos más complejos. WSCI logra esto mediante la definición de una capa en la parte alta del servicio web. Esta capa describe el comportamiento requerido de un servicio web relativo al intercambio de mensajes que soporta.

WSCI soporta los siguientes requisitos clave para un intercambio de mensajes coreografiado, dinámico y duradero:

- **Coreografía de mensaje**. WSCI describe el orden en el que se pueden enviar o recibir los mensajes en un determinado intercambio, las reglas que rigen dicha ordenación, los límites del intercambio de mensajes (cuando empieza y cuando finaliza), etc. Sin embargo, WSCI no define cómo se gestiona la coreografía del servicio web internamente.
- **Límites de transacción y compensación**. WSCI describe qué operaciones son realizadas de forma transaccional e informa a otros participantes de la capacidad de dicho servicio web para realizar esas operaciones. Esa capacidad también permite al servicio web unirse eventualmente a una transacción distribuida con otros servicios con los que interactúa.
- WSCI no define un protocolo de dos fases sobre la web o cómo se realizan las transacciones a través de servicios web. En su lugar, permite a un servicio web informar cuando es capaz de gestionar operaciones de forma transaccional para definir de manera precisa los límites de transacción y el comportamiento de compensación externamente observable.
- **Tratamiento de excepciones**. Con ello se define cómo reacciona el servicio web cuando suceden hechos inesperados, proporcionando una descripción de patrones de comportamiento alternativos, aunque no define un mecanismo por el cual un servicio web reaccione y gestione las situaciones excepcionales. De nuevo permite a un servicio web describir su comportamiento observable cuando se encuentra relacionado con desviaciones en su comportamiento habitual.
- **Gestión de hilos**. Con ello se describe si un servicio web es capaz de gestionar múltiples conversaciones (basadas en el mismo intercambio de mensajes) con el mismo o diferentes interlocutores y cómo lo lleva a cabo. De la misma forma, describe las relaciones entre los diferentes propietarios de mensajes en el mismo intercambio.
- Nuevamente, no se define la forma en la que el servicio web es capaz de gestionar las conversaciones de forma concurrente y permite afirmar los elementos observables en el intercambio de mensajes. Más allá, WSCI permite al servicio web describir las relaciones entre las partes de diferentes mensajes que colectivamente garantizan la consistencia del intercambio de mensajes.
- **Propiedades y selectores**. Describe los elementos que influencia el comportamiento observable de un servicio web. A pesar de que WSCI no es un lenguaje ejecutable que provea la semántica para las variables utilizadas para automatizarlo, permite a un servicio web afirmar cuales son los elementos relevantes en el intercambio de mensajes que ejercen influencia en el comportamiento observable en cualquier momento dado.
- **Conectores**. Describe como las operaciones realizadas por diferentes servicios web actúan en el mismo intercambio de mensajes. WSCI permite el mapeado de operaciones ''consumidoras'' desde un servicio web a operaciones ''productoras'' desde otro servicio web con el objetivo de construir un modelo global de intercambio en el que no exista ambigüedad. Desde el punto de vista de WSCI Global Molde también es posible describir cómo las interfaces de diferentes servicios que participan

en un mismo intercambio pueden ser unidas para construir un modelo de interacciones end-to-end.

- **Contexto operacional**. Describe cómo se comporta el servicio web en el contexto de diferentes intercambios de mensajes. Diferentes interfaces WSCI pueden ser asociadas con diferentes contextos operacionales en los que participa el servicio web. WSCI no define el comportamiento de un servicio web independientemente de cómo se utilice.
- **Participación dinámica**. Describe cómo se selecciona de forma dinámica la identidad del objetivo del servicio. Esta selección se basa en algún criterio que se conoce en tiempo de ejecución y que depende de la información descrita por la propia interfaz WSCI. Conociendo las propiedades utilizadas para identificar el servicio objetivo es posible comprende cómo, cambiando el valor de dichas propiedades, afecta el comportamiento del servicio correspondiente.

La definición de un mecanismo particular para identificar el servicio objetivo se encuentra fuera del alcance de WSCI y puede ser abordado por futuras especificaciones. Todo esto se puede resumir en que WSCI describe todos los elementos requeridos para proporcionar:

- La vista externa de cómo interactúan los servicios web con otros servicios en el contexto de un intercambio de mensajes, o lo que es lo mismo, cómo el servicio web es percibido para interactuar con el mundo exterior en un intercambio de mensajes.
- La vista de los intercambios de mensajes desde la perspectiva del servicio web, es decir, cómo el servicio web percibe el comportamiento del mundo exterior en el contexto de un intercambio de mensajes dado.

2.7.3.2.1 Estructura

La estructura que presenta el lenguaje se compone de los siguientes elementos:

- **Interfaz**: <interface> describe el flujo de mensajes que intercambia un servicio web en una conversación con otros servicios, es decir, las dependencias lógicas y temporales entre los mensajes intercambiados. Cada interfaz describe un escenario particular en el que interviene el servicio (un servicio puede tener varias interfaces WSCI).
- **Modelo**: Flujo global de mensajes entre una colección de servicios de conversan. Se proporciona una descripción de los mensajes intercambiados desde la perspectiva global de los participantes (colección de interfaces). <model> referencia interfaces individuales por su nombre, <connect> enlaces entre las operaciones de los participantes.
- **Actividad**: Unidad básica de comportamiento de un servicio (enviar o recibir un mensaje) Existen varios tipos de actividades:
    - Atómicas (corresponde con operaciones WSDL): el observador no puede consultar el estado intermedio del servicio durante su ejecución <action>
    - Complejas: involucra varias operaciones atómicas.

  <activity set> representa conjunto de actividades asociadas al contexto en el que se ejecutan, <complex activity> una actividad compleja que contiene uno o más <activity set> coreografiados.

- **Coreografía**: Descripción de las dependencias lógicas y temporales entre actividades. Existen una serio de elementos del lenguaje que se presentan a continuación:
    - <all> ejecuta un conjunto de actividades en un orden no secuencial (paralelo)
    - <choice> selecciona la ejecución de una actividad en función de un evento
    - <foreach> ejecuta un conjunto de actividades repetidas veces en función de una lista de elementos
    - <sequence> ejecuta todas las actividades en orden secuencial
    - <switch> selecciona la ejecución de una actividad en función de una o más condiciones expresadas en Xpath (Case y Default)
    - <while> ejecuta un conjunto de actividades repetidas veces en función de la evaluación de una condición en Xpath (bucle pre-test)
    - <until> ejecuta un conjunto de actividades repetidas veces en función de la evaluación de una condición en Xpath (bucle post-test)
- **Procesos**: Es un tipo especial de actividad (<activity set>) que crea su propio contexto y es etiquetado con un nombre, lo que permite la reutilización. Se puede instanciar por recepción de un mensaje o invocándolo explícitamente con Call, Spawn o Join.
- **Propiedades**: Asocian un nombre a un valor no tipado dentro del ámbito de un contexto (similar a las variables habitualmente presentes en lenguajes de programación)
- **Contexto**: Entorno en el que se ejecutan una colección de actividades, <context>. Comprende la declaración de las propiedades y procesos visibles para las actividades, excepciones y transacciones locales.
- **Correlación de mensajes**: Describe cómo las conversaciones son estructuradas y qué propiedades son intercambiadas para mantener la consistencia semántica de la conversación. Es además el mecanismo por medio del cual un mensaje se asocia a una conversación. Se representan como <correlation>, <correlate> e <instantation>.
- **Excepciones**: Asociadas a la ocurrencia de un evento, de un tiempo límite o un fallo. Una vez ocurridas ejecutan una actividad asociada y finaliza el contexto actual (<exception>)
- **Transacciones**: Permite describir una colección de actividades que se ejecutan como una única unidad de trabajo (<transaction>, <compensate>).

La siguiente figura muestra un ejemplo en el que se puede observar la forma en la que se utilizan algunos de los elementos comentados anteriormente:

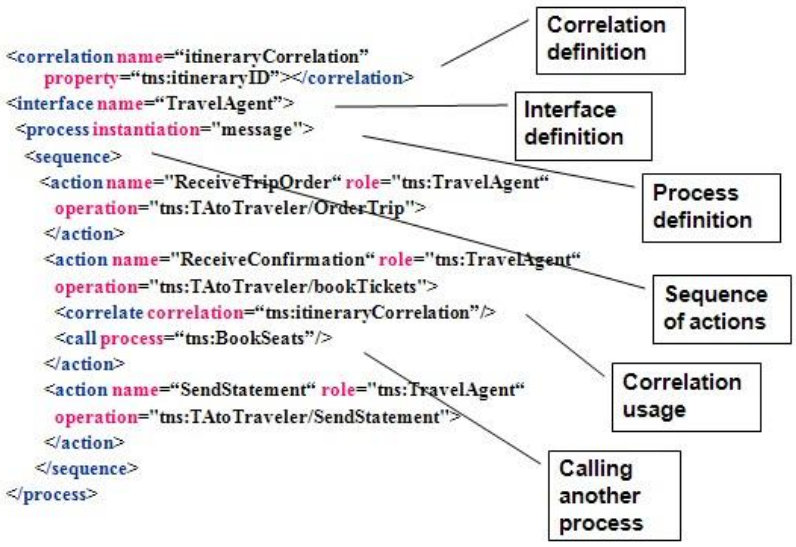


Figura 28. Ejemplo de utilización de elementos WSCI

2.7.3.2.2 Ventajas e inconvenientes

A continuación se presentan las principales ventajas e inconvenientes derivados del empleo de WSCI:

- Ventajas:
  1. WSCI está basado en XML.
  2. Las declaraciones WSCI pueden ser fácilmente incorporadas en la especificación WSDL como definiciones.
- Inconvenientes
  1. Las herramientas para la edición de las especificaciones WSCI son muy rudimentarias
  2. Existe el sentimiento de que la especificación propuesta por WSCI puede resultar útil pero no existen aún suficientes iniciativas para:
     - Reforzar las afirmaciones de WSCI como requisitos para la implementación de servicio web.
     - Generar modelos de análisis (por ejemplo para simulación) basadas en especificaciones WSCI.

2.7.3.3 BPEL

BPEL, en inglés Business Process Execution Language, es un lenguaje de ejecución estándar que surge con la finalidad de tener un patrón de diseño relacionado a la orquestación de Servicios Web. El lenguaje fue originalmente escrito por BEA Systems, IBM y Microsof, encontrándose ahora en el seno de OASIS (Organization for the Advancement of Structured Information Standards). El lenguaje BPEL permite definir la lógica de orquestación entre los diferentes servicios web. La entidad encargada de la orquestación es el director, que no es otro que el motor de ejecución BPEL (Fu, Bultan, & Su, 2004).

Se desarrolló a partir de WSFL y XLANG, ambos lenguajes basados en XML y orientados a la descripción de servicios web. Básicamente consiste en un lenguaje basado en XML diseñado para el control centralizado de la invocación de diferentes servicios web, con cierta lógica de negocio añadida que ayudan a la programación en gran escala.

La programación en gran escala generalmente se refiere a desarrollo de software de gran tamaño que involucran grandes procesos de desarrollo, evolución y mantenimiento. Por otro lado la programación detallada se refiere a la construcción de componentes software pequeños y autónomos. El desarrollo de BPEL nace de la necesidad de manejar lenguajes distintos entre la programación a gran escala y la programación detallada, ya que en su esencia ambos tipos de desarrollo requieren de distintos grados de comunicación con otros servicios.

El lenguaje BPEL permite lo siguiente:

- Definir procesos de negocio que interactúan con entidades externas mediante operaciones de un servicio web definidas usando WSDL 1.1 y que se manifiestan a sí mismas como un servicio web.
- Definir procesos de negocio utilizando un lenguaje basado en XML. No definir una interpretación gráfica de procesos o proveer de una metodología de diseño en particular.
- Definir una serie de conceptos de orquestación de servicios web que pretenden ser usados por vistas internas o externas de un proceso de negocio.
- Proveer de sistemas de control jerárquicos y estilo gráfico que permitan que su uso sea fusionado lo más ininterrumpido posible. Esto reduciría la fragmentación del espacio del modelado de procesos.
- Proveer de funciones de manipulación de datos simples requeridas para definir datos de procesos y flujos de control.
- Soportar un método de identificación de instancias de procesos que permita la definición de identificadores de instancias.
- Usar servicios web como modelo para la descomposición y ensamblaje de procesos.

El lenguaje BPEL se compone de los siguientes elementos básicos:

- **Definición**: es en este elemento donde se realiza la definición del proceso a modelar.

<process name=''tuproceso”> actividad </process>

- **Partners**: en BPEL todo servicio web involucrado en el proceso se modela como un ''Partner Link'' cuya estructura es definida como unPartnerLinkType en el archivo WDL propio del proceso. En donde el PartnerLinkType especifica el rol y tipo del partner involucrado. Su definición es la siguiente:

<partnerLinks> <partnerLink name=''uservicio'' partnerLinkType=''qname'' myrole=''ncname'' partnerRole=''ncname''> </partnerLink> </partnerLinks>

- **Variables**: estas se definen en su estructura de la siguiente manera:

<variables> <variable name=''ncname'' messageType=''qname'' type=''qname'' element=''qname''/> <variables>

Existen tres tipos básicos de variables en BPEL estos son: WSDL Message Type, XML Schema Type y XML Schema Element.

- **Asignación**: así como se puede ''guardar'' el estado de un determinado mensaje, se puede entonces transmitir esos valores de una variable a otra. Su definición es de la siguiente manera:

<assign> <copy> <from variable=''ncname'' part=''ncname''/> <to variable=''ncname'' part=''ncname''/> </copy> </assign>

- **Actividades Básicas**: en BPEL se pueden modelar las siguientes actividades para ser llevadas a cabo en un determinado proceso:
  1. Invoke
  2. Receive
  3. Reply
  4. Signaling
  5. Faults
  6. Waiting
  7. Doing
  8. Nothing

- **Actividades Estructuradas**: estas actividades están diseñadas con la finalidad de brindar de una "estructura" al proceso que será modelado. Éstas son:

  1. Actividades de control de secuencia básica, dadas por: Sequence,Switch y While.
  2. Actividades de Sincronización dadas por la definición de: Flow.
  3. Elección basada en información provista por entes externos definidas por: Pick

2.7.3.4 WS-CDL vs BPEL

Es inevitable observar algunas diferencias existentes entre BPEL y WS-CDL. Para esclarecer las posibles ambigüedades, se presentan a continuación las principales diferencias existentes entre ambos lenguajes:

- WS-CDL proporciona una definición de los formatos información que están siendo intercambiados a todos los participantes, mientras que BPEL proporciona esa información para un único participante.
- WS-CDL proporciona intercambio de mensajes global entre participantes sin un punto de vista específico. BPEL proporciona el intercambio de mensajes desde el punto de vista de un participante.
- WS-CDL proporciona reglas ''reactivas'' que son utilizadas por cada participante para computar el estado de la coreografía e inferir que intercambio de información sucederá o puede suceder a continuación. BPEL especifica reglas ''activas'' que son ejecutadas para inferir que hacer a continuación, una vez que la regla es procesada, la orquestación en tiempo de ejecución ejecuta la actividad o actividades correspondientes.

Además, WS-CDL ofrece un modelo para especificar un entendimiento común de un intercambio de mensajes (si un intercambio de mensajes se lleva a cabo, se acuerda que ambos estarán en un determinado estado, por ejemplo). BPEL no posee dicho concepto.

Por tanto y como conclusión, se puede decir que WS-CDL es el último esfuerzo del W3C Choreography Working Group en relación a las coreografías entre servicios web. Está construida con estándar de servicios web y establece distinciones de los roles que los participantes pueden adquirir. Se ha definido una gramática XML Schema pero no existen implementaciones o arquitecturas desarrolladas y disponibles para probarla. Sin embargo, no posee semánticas formales, no existe posibilidad de detectar deadlocks y no es posible retomar una conversación pasada.

2.8 Sumario

El estado del arte presentado proporciona una visión detallada de los principales conceptos que participan de la investigación y constituyen un conjunto tecnologías maduras y adecuadas para alcanzar los objetivos propuestos.

Una vez desarrollada toda la información relativa al estado del arte del trabajo, comienza un periodo de análisis de la misma que permitirá seleccionar los elementos y tecnologías más apropiados dentro de los campos estudiados. Es por ello que se hace necesaria una reflexión acerca de cuáles son las necesidades reales de la investigación y qué aspectos se adaptan mejor para cubrirlas en su totalidad.

Las tecnologías semánticas son de vital importancia en la investigación. Las ontologías permitirán representar el conocimiento de las aplicaciones de la plataforma de Cloud

Computing y el lenguaje desarrollado se servirá de la semántica propia de alguno de los lenguajes clave de estas tecnologías para su funcionamiento.

Los servicios web aportan el componente de comunicación entre las aplicaciones. Bien es cierto, que el lenguaje será el encargado de gestionar los parámetros de esta comunicación, pero se hace necesaria la presencia de un sistema que permita, propiamente, el intercambio de información. En este aspecto, la participación de los servicios web semánticos no se observa de forma clara y completamente necesaria ya que el lenguaje es el que se encarga de dotar de esa inteligencia previamente mencionada y de la automatización de procesos, actividades principales que caracterizan a los servicios web semánticos. De esta manera, el lenguaje y sus aportaciones semánticas sustituyen las características propias de los servicios web semánticos y las hacen más específicas y adecuadas para el entorno en el que van a funcionar.

Por último, el paradigma del Cloud Computing aporta el entorno en el que se desarrollarán estas actividades. Sus características lo hacen ideal para cumplir con los objetivos de la investigación, permitiendo establecer un entorno cerrado y controlado y formando un conjunto de tecnologías con una presencia cada vez mayor en todo tipo de sectores y cuya evolución se espera que continúe debido al éxito que está cosechando.

Con todo esto, se puede concluir que las tecnologías seleccionadas resultan adecuadas para afrontar los problemas expuestos y que su utilización conjunta en este trabajo supone una aportación original y novedosa al estado del arte. Además, los resultados que se obtengan pueden contribuir a incrementar el interés en el área y permitir la apertura de nuevas líneas de investigación que completen y complementen las ya existentes, situando este trabajo en la primera línea de los distintos ámbitos objetos de estudio.

3 Propuesta de solución

En este capítulo se plantean las hipótesis que forman la base de esta investigación. A partir de estas hipótesis, la solución propuesta es un lenguaje común o estándar dotado de semántica que con la ayuda de una ontología será capaz de capturar el conocimiento de las aplicaciones alojadas en una estructura de computación en nube. De esta manera, un lenguaje basado en ontologías se plantea como una solución válida para solucionar el problema de la interoperabilidad entre aplicaciones. Esto es debido a que mediante sus propiedades será posible conocer la información que poseen todas ellas y almacenarla para así poder usarla, interconectando unas aplicaciones con otras con el fin de producir un intercambio información de forma automática cuando sea preciso.

3.1 Hipótesis de Investigación

El concepto de hipótesis posee múltiples definiciones y acepciones. Es un término que se maneja en diferentes disciplinas con distintos puntos de vista en todas ellas, si bien siempre se trata a grandes rasgos de lo que se puede definir como una suposición, una idea basada en información previa que puede ser verdadera o no.

En el ámbito científico las hipótesis se definen como proposiciones aceptables formuladas mediante la recopilación de información y datos que sirven para responder de forma alternativa a un problema con base científica. Se presenta como el elemento fundamental de un proceso de investigación, al que condicionan y orientan para permitir llegar a una serie de conclusiones concretas que permitan establecer su veracidad o falsedad.

Para la investigación desarrollada, las hipótesis se centran en la posibilidad de creación de un lenguaje basado en tecnologías propias de la Web Semántica que permita solucionar el problema de la interoperabilidad entre aplicaciones en entornos de Cloud Computing. Las proposiciones planteadas realizan afirmaciones fundamentadas que en caso de probarse como ciertas (la validación de las hipótesis se realizará en el Capítulo 5), supondrían la consecución de todos los objetivos marcados desde el inicio de la investigación así como un avance significativo en el estudio de la interoperabilidad.

Para ello, se plantean a continuación las hipótesis de las que parte la presente tesis doctoral y que marcan el proceso de investigación realizado. Estas hipótesis se dividirán en principales y secundarias, de forma que la demostración de las secundarias supondrá la validación de la principal que engloba a todas ellas:

1. Un lenguaje declarativo semántico basado en ontologías soluciona el problema de la interoperabilidad entre aplicaciones dentro de un entorno de Cloud Computing. Esta hipótesis lleva implícita la consecución de una serie de hipótesis secundarias que han de ser cumplidas para validar la hipótesis. Las hipótesis secundarias se presentan a continuación:

    1.1. Es posible desarrollar un lenguaje basado en ontologías.
    1.2. Un lenguaje basado en tecnologías semánticas permite solucionar el problema de la heterogeneidad e incompatibilidad entre aplicaciones.

2. Dotar de semántica a un lenguaje confiere propiedades ventajosas para la interrelación entre aplicaciones y la localización de información. Al igual que la hipótesis anterior, esta será validada a partir de la validación de sus hipótesis secundarias:

    2.1. Utilizar un medio de representación del conocimiento semántico proporciona ventajas a la hora de acceder, modificar y consultar la información relativa al intercambio de datos entre dos o varias aplicaciones en entornos de computación en nube.
    2.2. Un lenguaje basado en tecnologías semánticas permite a las aplicaciones intercambiar información de manera automática, sin precisar interacción por parte del usuario.

3. Es posible que varias aplicaciones heterogéneas intercambien información en entornos cerrados como el proporcionado por una estructura de computación en nube. De manera análoga a las anteriores, esta hipótesis viene acompañada de una serie de hipótesis secundarias que deberán cumplirse para poder validar la hipótesis:

    3.1. Un entorno cerrado permite el intercambio de información entre aplicaciones heterogéneas.
    3.2. Los entornos de Cloud Computing soportan la interacción automática de las aplicaciones alojadas en ellos.
    3.3. La utilización de un estándar o medio de comunicación común permite reducir el número de transacciones durante los procesos de intercambio de información.

3.2 Estudio preliminar de soluciones

Esta investigación hace uso de las tecnologías y conceptos estudiados a lo largo del estado del arte para resolver el problema de la integración a nivel de aplicación y, por tanto, de la interoperabilidad entre aplicaciones, afrontando los retos y problemas explicados en la sección 1.3 y presentando un enfoque original basado en tecnologías semánticas, lenguajes y Cloud Computing. Para ello, se ha diseñado CARL (Complex Application and Representation

Language), un lenguaje declarativo basado en OWL-DL (McGuinness & van Harmelen, 2004). Este lenguaje posee una sintaxis propia y propone una semántica basada en Lógica Descriptiva (Baader, 2003) capaz de capturar el contenido de las aplicaciones subidas a una plataforma de computación en nube y también de comprender los requisitos necesarios a nivel de datos para establecer un nexo entre aplicaciones que permitan la comunicación entre ellos aprovechando al máximo las características proporcionadas por este paradigma. Resumiendo, este trabajo se enfrenta a uno de los mayores problemas que presenta la Web como es la integración de aplicaciones heterogéneas, un problema con el que la Web Semántica lleva luchando desde sus comienzos.

3.2.1 Lenguajes declarativos

Un lenguaje declarativo se puede definir como el lenguaje de programación basado en las Matemáticas y la Lógica de los lenguajes imperativos cercanos al razonamiento humano debido a su capacidad de indicar qué tarea o acción debe realizarse, sin señalar la manera o el cómo debe llevarse a cabo (Dekel, Cohen, & Porat, 2003). En este sentido, los lenguajes declarativos proporcionan un enfoque interesante en el desarrollo de diversos paradigmas y sistemas, como se refleja en (Hanus, 2007).

A menudo se hace complicado establecer la diferencia existente entre lenguajes declarativos e imperativos y es un aspecto que debe explicarse claramente para comprender de forma correcta las decisiones tomadas. La principal diferencia reside en que un lenguaje imperativo describe el algoritmo o la secuencia de pasos que se deben realizar a la hora de resolver un problema. Por otro lado, el enfoque de los lenguajes declarativos se centra más en la descripción del problema a solucionar, sin especificar los pasos necesarios para conseguirlo, que vendrán dados por mecanismos internos o procesos de inferencia realizados a partir de la descripción previa del problema, sistemas que no existen en los lenguajes declarativos.

Los lenguajes declarativos se han utilizado en diversos ámbitos y áreas debido a su flexibilidad y capacidad de adaptación a diferentes entornos. Los trabajos previamente realizados en sistemas de Cloud Computing son muy limitados pero es interesante señalar que en (Alvaro et al., 2010) se realiza un estudio en el que se propone un modelo de programación declarativa en entornos distribuidos, como son los de computación en nube.

Los trabajos más relevantes en este campo se pueden encontrar en aproximaciones para la seguridad de la Web (Abadi & Loo, 2007) o para la estructuración de información como en (Lakshmanan, Sadri, & Subramanian, 1996) donde un sencillo sistema lógico basado en lenguajes declarativos es capaz de devolver información desde documentos HTML alojados en la Web. Además, también es posible encontrar algunos enfoques para negocios como por ejemplo en (Pesic & van der Aalst, 2006). En este caso, los autores proponen un enfoque declarativo para alcanzar el paradigma fundamental para la gestión flexible de procesos de negocio dinámicos debido a la propiedad de estos modelos para la especificación de qué se debe hacer en lugar de cómo, facilitando el trabajo a los usuarios que en la mayoría de los casos no tienen la capacidad de completar esas tareas. Más cercano al lenguaje que se desarrollará durante el transcurso de esta tesis se encuentra (H. Panetto, Berio, Benali,

Boudjlida, & Petit, 2004) que afronta el problema de la interoperabilidad de modelos de negocio mediante la creación de un lenguaje de modelización que sirva como lenguaje intermedio para la comunicación entre las diferentes entidades que toman parte en el proceso.

3.2.2 Conjunto de prerrequisitos

En esta sección se presenta un estudio preliminar de las entidades y conjuntos de prerrequisitos que deben estar presentes y ser cumplidos por el lenguaje CARL de forma que se abarquen los objetivos finales de la tesis doctoral.

Al introducir requisitos para cualquier software, hay dos aspectos que se deben tomar en consideración:

1. Qué engloban esos requisitos y cómo las tecnologías del actual estado del arte cubren esos requisitos. El primer paso que se debe realizar es otorgar de forma racional una importancia a esos requisitos mediante el análisis de su contexto y sus bases.
2. Por qué esos requisitos no se encuentran cubiertos con los actuales trabajos en el área.

A continuación se detallan los prerrequisitos que debe cubrir el lenguaje objeto de la investigación:

1. **Aplicación Compleja (CA)**. El primer conjunto de requisitos que se debe establecer es el de una descripción conceptual de lo que se llamará ''Aplicación Compleja''. Se considera una CA una de las entidades involucradas en la plataforma de aplicaciones y se presenta como la principal del modelo conceptual. Debe hacerse explícita desde un punto de vista conceptual y también debe verse reflejada en la sintaxis interna.
2. **Conformidad Conceptual (CC)**. Un modelo conceptual es un conjunto de conceptos y relaciones que describe un dominio específico. El modelado conceptual es un área de investigación que ha sido objeto de una intensa actividad investigadora, especialmente por el área de la Representación de Conocimiento (Huhns & Singh, 1998). El modelo conceptual que se presentará consistirá en un conjunto de conceptos, relaciones e instancias.
3. **Fundamentos Formales (FF).** En este lenguaje, el FF está basado en semánticas formales que proporcionan significado a los programas de ordenador. Este significado permite el razonamiento sobre esos programas, basados en propiedades matemáticas de la semántica aplicada. El razonamiento es el proceso de extraer conclusiones a partir de hechos.

   Se pueden definir cuatro estilos principales de razonamiento formal sobre sistemas computacionales centrados en proporcionar semántica (significado) a los programas:

- *Semántica operacional*. Un programa de ordenador se modela como una ejecución de una máquina abstracta. Un estado de dicha máquina se define como una evaluación de variables. Instrucciones simples representan transiciones entre estados (McCarthy, 1961).
- *Semántica denotacional*. Los programas son representados como una función entre la entrada y la salida (Scott & Strachey, 1971).
- *Semántica axiomática*: Para probar si un programa es correcto se utilizan métodos de prueba. Las notaciones centrales son aserciones de programa, tripletas de prueba consistentes en precondiciones, declaraciones del programa, post-condiciones e invariantes (Hoare, 1978).
- *Semantica estructural*. Se basan en lógicas para la descripción (Donini, Lenzerini, Nardi, & Schaerf, 1997) lo que permite operaciones de inferencia y el procesamiento de información contenida en las aplicaciones. Dentro de este grupo destaca OWL (McGuinness & van Harmelen, 2004), un lenguaje de marcado ideado para publicar y compartir datos mediante ontologías. El uso de Description Logics (DL) le confiere un gran potencial.

Para la ejecución de la semántica se utilizará la semántica estructural basada en Lógica de Descripción. Esta decisión ha sido tomada en base a las características que este tipo de semántica confiere y a su propiedad de decibilidad, que permite establecer un método lógico y finito para alcanzar una solución válida. De manera general, se pueden contemplar además que si la especificación está escrita en un lenguaje lógico que pueda usar inferencia es factible extraer consecuencias a partir de esa propia especificación. Esto se presenta como una gran ventaja a la hora de utilizar métodos formales. Utilizando esta característica, las propiedades de la especificación que no se encuentran explícitamente representadas, pueden ser probadas. De esta manera, comportamientos y propiedades no visibles del sistema, pueden ser predichos y probados sin necesidad de ser implementadas, como por ejemplo, situaciones de deadlock o errores.

Además, el uso de OWL conlleva numerosas ventajas para alcanzar los objetivos propuestos. OWL está pensado para ser usado cuando la información contenida en los documentos necesita ser procesada por las aplicaciones, al contrario que en las situaciones donde el contenido sólo necesita ser presentado a los humanos. OWL puede ser usado para representar explícitamente el significado de términos en vocabularios y las relaciones entre esos términos mediante ontologías. Tiene mayor capacidad para expresar significado y semántica que XML, RDF, y RDF-S, y, de este modo, OWL va más allá de estos lenguajes en su capacidad para representar contenido interpretable por un ordenador en la Web. OWL añade, además, vocabulario para describir propiedades y clases: entre otros, relaciones entre clases, cardinalidad, igualdad, más tipos de propiedades, características de propiedades, clases enumeradas, etc. Lo que lo hace más que idóneo para el caso del que se ocupa esta investigación. Otras fortalezas de OWL son las siguientes:

- Es compartible (Sanin, Szczerbicki, & Toro, n.d.).
- Modificable en el tiempo.
- Interoperable.
- Con detección de inconsistencias.
- Posee balance entre expresividad y complejidad.
- Facilidad de uso.
- Es compatible con estándares existentes.

Por tanto, el requisito aquí presentado es el de emplear semántica en el diseño del lenguaje. La razón de ser de esta ejecución semántica formal posee doble vertiente: comprobar automáticamente las propiedades del sistema y ayudar a los desarrolladores a comprender la especificación.

3.3 Complex Application and Representation Language (CARL)

CARL es el lenguaje semántico propuesto para solucionar el problema planteado en la investigación. Su nombre refleja su capacidad para representar y capturar la información relativa a las aplicaciones complejas (concepto definido en la sección 3.2.2) que se alojan en la estructura de computación en nube. Una vez representada esa información, ésta es almacenada para que posteriormente se pueda realizar el proceso de comunicación entre las diferentes aplicaciones que lo necesiten.

Esta sección contiene los diferentes aspectos de diseño necesarios para el modelado del lenguaje CARL, una explicación del modelo conceptual, su sintaxis y la ontología que ha motivado el desarrollo de este trabajo.

3.3.1 Modelo Conceptual

El modelo conceptual representa los diferentes elementos que participan en el sistema de información. En este caso, el modelo conceptual se refiere a la ontología diseñada para capturar el conocimiento del contexto necesario para cubrir los requisitos previamente mencionados y definir las tareas a realizar.

La

Figura 29 muestra una aproximación de alto nivel de los conceptos que se representarán en la ontología CARL y que permiten el modelado de la interoperabilidad entre aplicaciones heterogéneas:

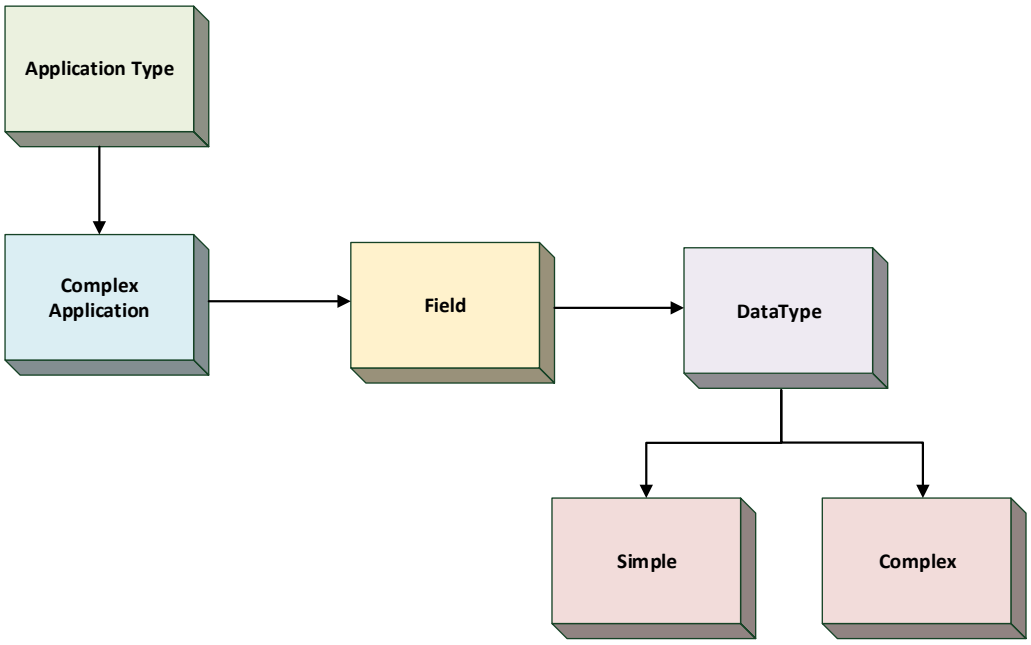


Figura 29. Modelo Conceptual CARL

A continuación se proporciona una breve descripción de cada uno de los conceptos reflejados en la figura anterior:

- ApplicationType (AT). Es la clase más general del modelo conceptual. Engloba y clasifica a las aplicaciones complejas en una serie de conjuntos según el ámbito o sector de las mismas. Gracias a esta entidad se podrá mejorar el proceso de búsqueda mediante procesos de inferencia lógica.
- Complex Application (CA). Esta es la clase que gestiona las aplicaciones complejas que pueden intercambiar información a través del lenguaje. Como se puede ver en la
- Figura 29 se encuentra conectado con la entidad *Field* que refleja que cada CA poseerá un conjunto de campos que la identificarán. Es importante recalcar que es posible realizar operaciones lógicas sobre las aplicaciones complejas y sus campos, utilizando el potencial de la ontología para incrementar la calidad del proceso y asegurar el correcto funcionamiento del sistema. Es precisamente en este punto donde se proporciona una de las razones para el uso de ontologías en lugar de otro tipo de representación de información.
- Field. Refleja los diferentes campos que componen e identifican las aplicaciones complejas. Esta entidad permitirá definir la información que necesita cada CA para su funcionamiento y por otro lado capturar el conocimiento de la misma para su posterior utilización en los procesos de interoperabilidad. A su vez, cada uno de los campos de las aplicaciones vendrá definido por un tipo de datos (DataType) que fijará el formato del mismo.

- DataType (DT). Esta clase modela los diferentes tipos de datos que pueden existir en el sistema y que serán aplicados a cada uno de los campos que componen una CA. Estos tipos de datos se corresponden con entradas y salidas de las aplicaciones y permiten a la ontología conocer los requisitos de esas aplicaciones de manera que pueda favorecer la comunicación entre ellas. Los DT se pueden dividir en dos subtipos diferentes: *Simple* y *Complex*.
- Simple. En este modelo se define como un dato simple aquellos que se corresponden con los tipos básicos tradicionales para la representación numérica (int, double, long, byte, short, float), de caracteres (char), booleanos (boolean) y de cadenas de caracteres (String). Es decir, se considerarán tipos de datos simples los tipos primitivos y las cadenas de caracteres.
- Complex. Los tipos de datos complejos serán aquellos que necesitan un procesamiento previo para su obtención. Son tipos predefinidos en el sistema que pueden ser utilizados por las aplicaciones para definir sus campos de una forma más precisa facilitando a la vez su posterior intercambio en los procesos de interoperabilidad. Ejemplos de estos casos pueden ser la definición del tipo "DiaSemana" o los formatos predefinidos para las fechas. Más adelante se proporcionará información más detallada sobre la utilización de este tipo de datos (ver sección 3.3.3).

Con la definición de estos conceptos, los principales requisitos se encuentran cubiertos y es posible afirmar que se pueden alcanzar los objetivos establecidos para el lenguaje.

3.3.2 **Diseño de la ontología (CARL Ontology)**

La ontología diseñada tiene como objetivo representar y capturar el conocimiento contenido en las aplicaciones que se sitúen en la estructura de computación en nube. Este conocimiento consiste en la definición de los diferentes campos de los que se compone cada aplicación y el tipo de datos asociado a los mismos. De esta manera la ontología poseerá toda la información de las aplicaciones y servirá de base para realizar las conversiones de datos pertinentes en los procesos de intercambio de información, facilitando la interoperabilidad mediante la compatibilidad de formatos.

Existen determinados aspectos que justifican el empleo de ontologías como herramientas para la representación de información y que deben ser tenidos en cuenta para el diseño de la misma que permite afrontar los problemas expuestos:

- El primero de los motivos de su utilización es el de mejorar los procesos de búsqueda de aplicaciones. Mediante la interrelación de conceptos y el establecimiento de jerarquías, las ontologías son capaces de proporcionar una mayor cantidad de información relevante a través del uso de inferencia lógica. Este tipo de inferencia se denomina subsunción y juega un papel fundamental, ya que estableciendo conceptos y subconceptos se pueden hacer explícitos datos a priori ocultos. De esta manera las

ontologías permiten realizar procesos de inferencia que serían inviables mediante otros tipos de representación del conocimiento. A continuación se presenta un ejemplo de aplicación de subsunción en los procesos de búsqueda que clarifica la gran utilidad de las ontologías en este aspecto:

  - Un usuario desea contratar un vuelo y para ello realiza una búsqueda en el sistema de la aplicación de Iberia. Siguiendo el modelo conceptual diseñado, cada aplicación debe estar englobada por una categoría superior que indique el tipo de aplicación de la que se trata. Al introducir el criterio de búsqueda la ontología no sólo devolverá como resultado la aplicación de Iberia que buscaba el usuario, sino que también le proporcionará información adicional sobre otras compañías aéreas o agencias de viajes que pueden ser de su interés a la hora de contratar el vuelo que desea. Esta opción resulta interesante en muchos casos, ya que en una gran mayoría de las situaciones los usuarios no conocen todas las opciones de las que disponen para realizar un mismo trámite o consumir un servicio.

- El segundo aspecto principal del uso de ontologías para el problema que se desea resolver es el de la posibilidad de utilizar SPARQL como lenguaje de consultas, ya que está basado en RDF. SPARQL es una tecnología clave en el desarrollo de la Web Semántica que permite extraer información precisa mediante consultas lógicas, reduciendo así la complejidad de la propia consulta.

Mediante la presentación de estas posibilidades abiertas por las ontologías, queda justificada su utilización y adecuación para la resolución del problema de la interoperabilidad en el marco presentado.

A lo largo de la siguiente sección se exponen los diferentes pasos o fases seguidas para el diseño y construcción modular de la ontología mencionada. Como se ha comentado, se encuentra basada en el modelo conceptual desarrollado y es capaz de capturar el conocimiento de las aplicaciones para habilitar los procesos de intercambio de información en el sistema.

3.3.2.1 Primera fase

La construcción de la ontología se ha diseñado de manera modular con el fin de justificar cada una de las decisiones tomadas y en función de las necesidades del sistema para ajustarse al modelo conceptual planteado en la sección 3.3.1.

Para la representación del conjunto de aplicaciones existentes, se creó la clase “ComplexApp”, concepto que representará cada una de las aplicaciones que se suban a la infraestructura de computación en nube creada.

Posteriormente se analizó la forma de representar el conjunto de campos de los que está compuesta una aplicación. Se planteó la posibilidad de crear una superclase que englobaría dos clases:

- Field: Representaría cada uno de los campos que poseerá la aplicación en cuestión.
- Type1: Contendrá el tipo de datos de cada campo para la aplicación dada. Este campo se ha nombrado en la ontología como “Type1” por ser “Type” palabra reservada.

La relación de Field → Type1 a nivel de individuos se diseñó inicialmente de forma directa, es decir, se creó una relación del tipo “(Field) Edad es de tipo (Type1) int”. A continuación se pudo comprobar que esta opción no era viable, ya que si hubiera dos campos en la misma aplicación que fueran del mismo tipo se hacía necesario crear dos individuos iguales o bien tener dos propiedades para relacionar los dos campos con el tipo.

Esta situación complicaba mucho el manejo de la ontología y por ello se creó una superclase para poder relacionar los campos y los tipos sin necesidad de que la relación fuera directamente entre ellos. Esta clase se llamó “FieldAndType” y es la que define el campo y el tipo. Aquí habrá tantos individuos de FieldAndTypes como combinaciones posibles puedan ser abarcadas por todas las aplicaciones.

Una vez definida la estructura de clases que tendría la ontología se creó el conjunto de propiedades que relacionarían unos individuos con otros de otras clases. En este punto se definió la relación que podrían tener las clases entre ellas:

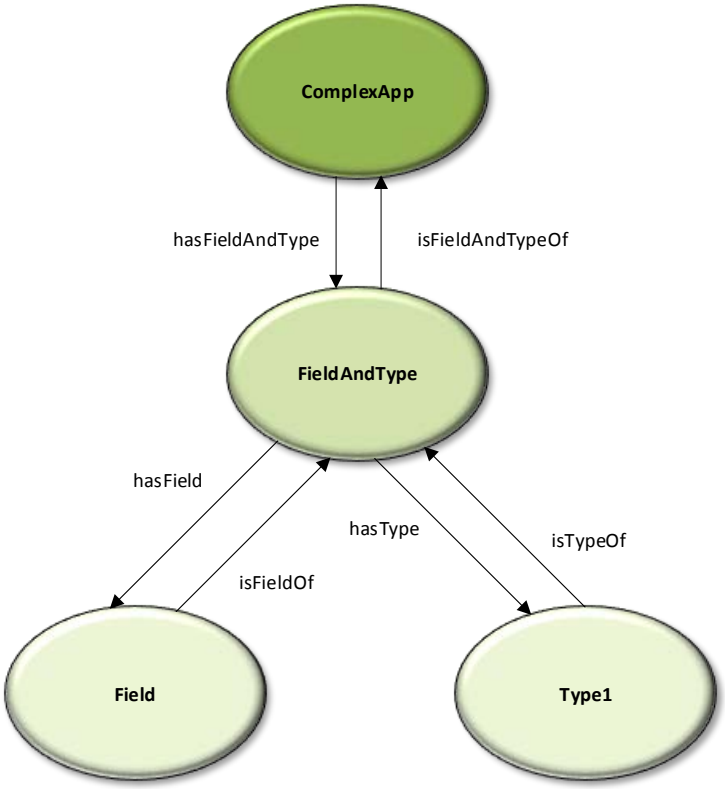


Figura 30. Primera fase del diseño de la ontología

3.3.2.2 Segunda fase

Una vez concluida la primera fase del diseño de la ontología, se observó que el diseño obtenido no permitía que dos aplicaciones tuvieran el mismo tipo de campos pero con nombres diferentes. Para contemplar esta situación, se debe añadir una nueva clase que consiga soportar el escenario planteado.

Tras analizar las opciones existentes en este aspecto, se realizó el nuevo diseño que se muestra en la Figura 31:

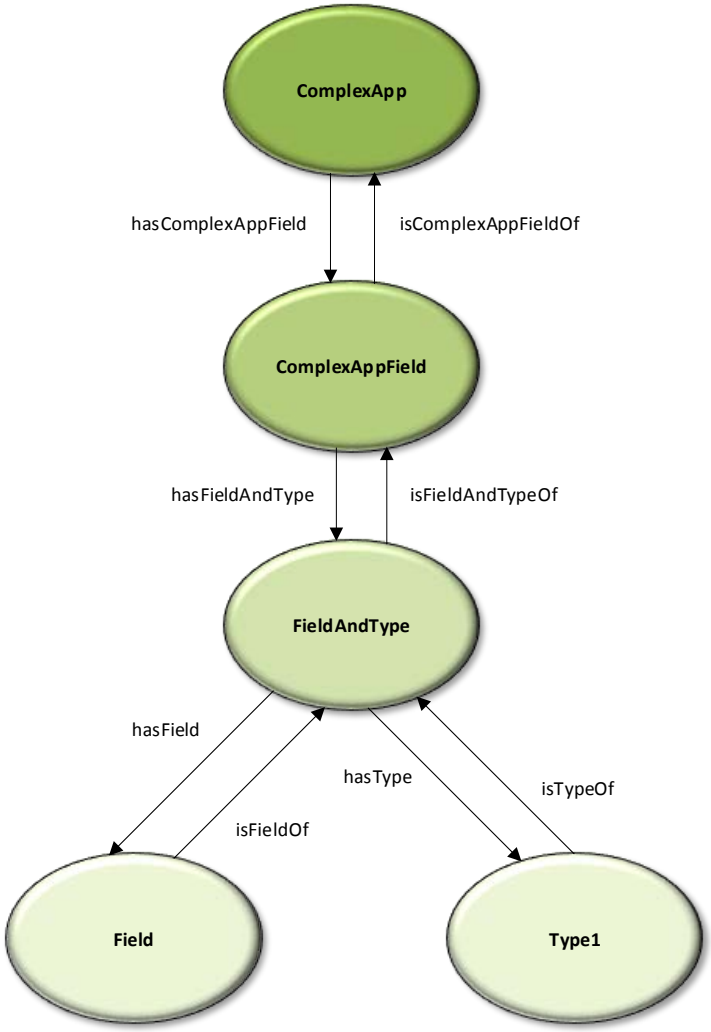


Figura 31. Segunda fase del diseño de la ontología

Como se puede observar, se ha añadido la una clase "ComplexAppField" que contiene los nombres de los campos de cada aplicación, permitiendo que existan campos iguales en las aplicaciones aunque posean nombres diferentes, cumpliendo uno de los aspectos más relevantes del sistema y solucionando el problema surgido.

En esta fase se analizó también la situación de la representación de las fechas, aspecto siempre problemático de afrontar. El inconveniente aparecía a la hora de identificar el formato de fecha, ya que con la configuración inicial la clase Type1 poseería únicamente individuos con formatos fijos (String, int, Data, etc.), haciendo imposible identificar el formato de fecha de manera correcta debido al gran número de posibilidades de representación que puede adoptar.

Desde esta perspectiva se planteó la posibilidad de crear una nueva clase que recogiera los diferentes formatos de fecha y relacionarla con la clase Type1. De esta forma se podría relacionar el individuo de la clase Fecha con cualquier individuo de la nueva clase, que contendría tantos individuos como formatos de fecha existen, como por ejemplo: yyyymmdd, yyyy-mm-dd, dd/mm/yyyy, etc. Cuando se planteó este cambio en la ontología se comprobó las tareas de administración podrían complicarse y dificultar las actualizaciones y manejo general de la misma. Por ello, la decisión tomada fue la de considerarlo como un tipo de dato complejo y preestablecer los formatos de fecha que se pueden utilizar. De esta manera se puede incluir un individuo por cada uno de estos formatos en la clase Type1, obteniendo tantos como formatos distintos existan.

3.3.2.3 Diseño final

Para el diseño final de la ontología se debían satisfacer los requisitos restantes que además permitiesen ser consistente con el modelo conceptual desarrollado. El primero de estos requerimientos era el de incluir la jerarquía en las aplicaciones de forma que permitiese definir cada una de las aplicaciones como un subconcepto de una clase mayor que las englobase y determinase su tipo. Por otro lado se encuentra la inclusión de una ontología del dominio que posteriormente proporcionará una funcionalidad clave para conseguir la interoperabilidad, estableciendo equivalencias entre los campos de la ontología CARL y los suyos, sirviendo de nexo para los diversos campos existentes en las aplicaciones.

Una vez contemplados estos aspectos, las clases que finalmente conforman el diseño de la ontología son:

- *AppType*: Es la clase padre de la jerarquía de la ontología. Conceptualmente representa el tipo de aplicación a la que pertenece cada una de las aplicaciones complejas.
- *ComplexApp*: Es la clase que representa a cada aplicación. Aquí se crearían tantos individuos como aplicaciones se incluyan en la infraestructura de Cloud Computing.
- *ComplexAppField*: Es la clase que contiene los nombres de los campos que tiene cada aplicación. Esta clase nos permite que una aplicación tenga el mismo tipo de campo que otra aplicación pero con nombres diferentes.
- *FieldAndType*: Esta clase es la que define el campo y el tipo. Habría tantas combinaciones como las que abarcasen todas las aplicaciones.
- *Field*: Esta clase contendría los individuos de los campos que puede tener una aplicación. Por ejemplo: Código Cliente, Fecha Nacimiento, Fecha de Alta, etc.

- *Type1*: Esta clase contendrá los individuos de los diferentes tipos posibles para los campos. Por ejemplo: String, Number, Date_yyyymmdd, Date_dd-mm-yyyy, etc.
- *DomainOntology*: Esta clase representa la unión de la ontología CARL con la ontología del dominio que se seleccione. La relación se establece con "Field" porque las equivalencias entre ambas se establecen a nivel del campo.

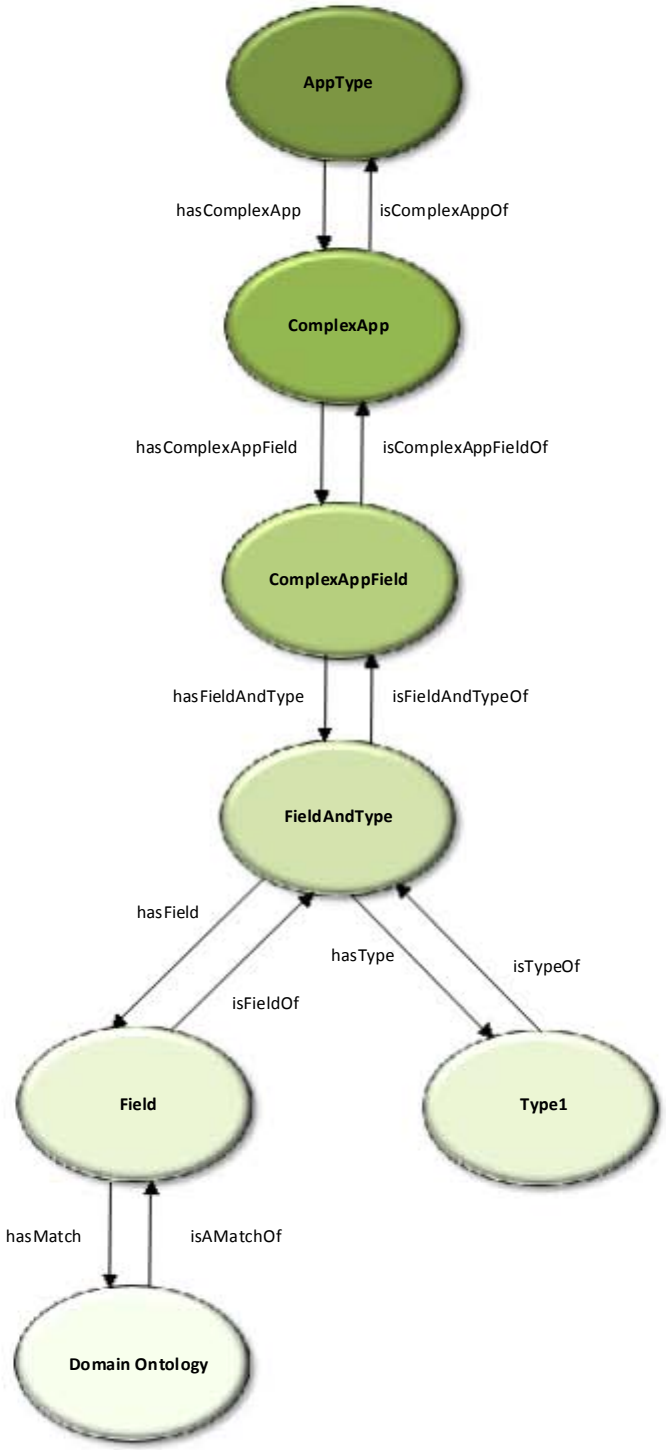


Figura 32. Diseño final de la ontología CARL

La Figura 32 muestra el diseño final de la ontología que se empleará para favorecer el intercambio de información entre aplicaciones de manera automática. La acción conjunta de esta ontología junto con la ontología del dominio y la definición del lenguaje necesario para capturar el conocimiento formará la base de los procesos de interoperabilidad.

Las fases por las que ha pasado el diseño de la ontología han mostrado el crecimiento de la misma a medida que se debían ir cumpliendo requisitos y funcionalidades para ofrecer los resultados esperados. Finalmente, todas las propiedades reflejadas cubren las necesidades del sistema y permiten, por tanto, afrontar los retos planteados.

3.3.3 **Sintaxis de CARL**

Como se ha comentado anteriormente, éste es un lenguaje basado en ontologías y en la semántica de OWL-DL que permite representar el conocimiento de las aplicaciones para poder establecer así un estándar de comunicación entre ellas.

Todo lenguaje precisa de una sintaxis que lo caracterice y permita realizar las operaciones necesarias para su funcionamiento. Para el caso del que se ocupa este trabajo, se ha decidido diseñar una sintaxis ajustada a las características del problema asociado a esta investigación, limitada al mínimo conjunto de operaciones que se deben realizar para establecer el proceso de comunicación deseado. De esta manera, se obtiene una mínima definición del lenguaje que permite usar la máxima semántica (proporcionada por OWL-DL).

A continuación se presentan los términos empleados por el lenguaje para la definición de los conceptos a representar en la ontología:

- **Define**: Inicia la definición de la aplicación que se va a representar. Se encarga de asignar el nombre a la aplicación que se está registrando, el cual se escribirá a continuación.
- **SubConceptOf**: Establece una relación de jerarquía. Una aplicación puede ser definida como un subconcepto de un concepto genérico superior habilitando así tareas de razonamiento en el sistema. Un concepto sólo podrá ser subconcepto de otro y los valores que podrá tomar se encontrarán predefinidos. Esto significa que la relación que se establece es única, biunívoca y de cardinalidad restringida.
- **Field**: Indica que lo que viene a continuación es el nombre de un campo existente en la aplicación.
- **Match**: Define concordancias entre clases (operaciones de alignment). Se especifica entre paréntesis el valor del campo con el que se realiza la unión. Especialmente útil para utilizarse con la ontología del dominio (todo lo referente a la ontología del dominio a utilizar se encuentra detallado en la sección 4.4).

- **Type**: Indica entre paréntesis el tipo de un determinado campo que ha sido definido previamente. Se utilizará sólo para tipos simples o primitivos correspondientes a valores numéricos (int, double, long, etc.), caracteres (char) y cadenas de caracteres (String).
- **TypeOf**: Este operador indica que se debe tomar el tipo correspondiente al del campo tomado de la ontología del dominio que se encuentra a continuación encerrado entre paréntesis. También se utilizará para opciones predefinidas que proporcionará el sistema para determinado tipo de campos y que podrán asignarse si se desea.
- **End**: Finaliza el proceso de asignación.

Con sólo la definición de estos operadores es posible representar toda la información relevante para el funcionamiento del sistema. A continuación, en la Figura 33, se presenta un ejemplo de cómo el lenguaje realiza la definición de una aplicación que se desea subir a la plataforma de Cloud Computing:

```
define IBERIA subConceptOf(CompañiaAerea);

    field  DNI_Pasajero   match(OntDom:Identificador)    type(int);
    field  nombre         match(OntDom:nombreCliente)    type(String);
    field  fechaNac       match(OntDom:fechaNacimiento)  typeOf(formatoFecha3);
    field  fechaVueloIda  match(OntDom:fechaEntrada)     typeOf(formatoFecha1);

end IBERIA;
```

Figura 33. Ejemplo definición aplicación con CARL

Siguiendo el modelo conceptual de la ontología diseñada se puede comprobar cómo la definición de estos aspectos permite representar toda la información que debe ser almacenada para posteriormente facilitar la consecución de la interoperabilidad.

En el ejemplo de la Figura 33 se encuentra la definición de la aplicación para la compañía aérea Iberia. Esta aplicación es subconcepto de la clase global "Compañía aérea" y consta de cuatro campos que poseen una correspondencia en la ontología del dominio a utilizar. Por otro lado, se hace necesario almacenar el tipo y el formato que poseen esos datos para poder posteriormente operar con ellos y realizar las transformaciones y equivalencias precisas. En el caso del ejemplo se extrae que:

- El campo "*DNI*" tiene un formato numérico.
- El campo "*Nombre*" se representa como una cadena de caracteres.
- El campo "*fechaNac*" posee el formato predefinido para fechas número 3.
- Por último el campo "*fechaVueloIda*" se corresponde con el tipo y formato predefinido para fechas número 1.

Como también se puede comprobar, todas las sentencias finalizan con un ";" que indica que dicha sentencia termina en ese punto. Para finalizar, la palabra reservada "end" indica que la definición de la CA ha finalizado.

Conocida esta información, se puede almacenar en la ontología y de esta manera saber cuál es el formato de los datos de cada campo en cada una de las aplicaciones existentes en la estructura de computación en nube. Así, al establecer una comunicación entre las aplicaciones, la ontología posee la información correspondiente y puede proporcionar el formato en el que se reciben los datos y en cuál se deben enviar. Realizando las transformaciones de datos precisas, el intercambio de información se puede realizar adecuadamente y de forma completamente automática, alcanzando los objetivos propuestos.

Relacionado con estos aspectos, resulta importante clarificar que la gestión de los propietarios de las aplicaciones así como de los usuarios que se suscriban a la misma, queda fuera de la influencia del lenguaje. CARL y su ontología se emplearán únicamente para la definición de aplicaciones y sus contenidos. Los propietarios y los usuarios finales son una parte fundamental del sistema global en el que CARL realizará sus actividades, pero al no precisar de esa información para su funcionamiento, sus datos serán almacenados en un componente adicional a la ontología de CARL.

Por último indicar que el archivo generado con la información a incluir en la ontología (como el mostrado en el ejemplo de la Figura 33), se genera con una extensión ".carl" que permite al sistema identificar la información que contiene.

3.4 Sumario

A lo largo de este capítulo se han expuesto las hipótesis que guiarán el proceso investigador así como la propuesta del lenguaje que permitirá alcanzar los objetivos planteados para, en último término, realizar la validación que permita demostrar dichas hipótesis. Por todo ello, la finalidad de esta sección es la de recoger un resumen con los aspectos más relevantes de la solución presentada y detallar cómo esta propuesta de solución consigue afrontar los retos de la investigación y alcanzar los objetivos marcados. Como se ha podido comprobar, la solución propuesta consiste en el desarrollo de un lenguaje basado en ontologías con una sintaxis propia y una semántica basada en lógica descriptiva (OWL-DL).

Una vez se fijaron los prerrequisitos que se debían cumplir así como el tipo de semántica que se debía emplear, se definieron los diferentes aspectos relacionados con el diseño de la ontología. El modelo conceptual de la ontología pretende ser lo más ajustado y restrictivo posible, de forma que se pueda representar todo el conocimiento necesario para asegurar el correcto funcionamiento del lenguaje, pero limitando el alcance de forma que se puedan evitar los problemas que hicieron fracasar los trabajos y aproximaciones anteriores. En función de este objetivo, se han tomado las decisiones de diseño que se consideran más adecuadas para la finalidad de la investigación.

Por otro lado, se ha realizado una mínima definición del lenguaje, a nivel de sintaxis, que permita emplear la máxima semántica. La definición de esta sintaxis viene dada por la necesidad de expresar las posibilidades del lenguaje. Se podría haber empleado también la propia sintaxis de OWL-DL, pero de esta manera se consigue hacer una definición más exacta y precisa del mínimo conjunto de operaciones necesarias para su funcionamiento. Para ello se han definido una serie de parámetros y operaciones capaces de abarcar toda la funcionalidad que debe proporcionar el lenguaje.

Con la definición de los diferentes aspectos que conforman el lenguaje, es posible detallar cómo estas especificaciones permiten afrontar los retos y objetivos que se plantearon al inicio de la investigación (ver sección 1.4.2). Por ello, se presenta a continuación un breve análisis de la forma en la que el diseño del lenguaje se enfrenta a los objetivos técnicos propuestos, dejando de lado los puntos de estado del arte y validación:

- El objetivo número 2 involucraba la definición de un modelo de intercambio de información o lenguaje que permitiese representar el conocimiento de las aplicaciones existentes en plataformas de Cloud Computing. La ontología diseñada a lo largo de este capítulo tiene como finalidad recoger ese conocimiento de las aplicaciones para que el lenguaje pueda hacer uso de él cuando sea necesario, haciendo frente al objetivo comentado.
- El objetivo número 3 presentaba la necesidad de añadir semántica al lenguaje para poder realizar operaciones lógicas e inferencia con el conocimiento capturado. Tras estudiar diferentes alternativas, el lenguaje diseñado hará uso de la semántica de OWL-DL, de forma que se pueda usar un razonador que permita realizar las operaciones de búsqueda e inferencia necesarias para completar el funcionamiento del lenguaje.
- El objetivo número 4 afronta la necesidad de establecer interacciones entre las diferentes aplicaciones existentes en el entorno de acción del lenguaje. Gracias a la capacidad del lenguaje de hacer uso del conocimiento representado en la ontología, la cual contiene información de las aplicaciones, se puede conocer la situación de cada una de ellas y emplear esos datos para establecer procesos de comunicación. Además, los operadores definidos para el lenguaje facilitan esta tarea, y su funcionamiento junto con la acción de servicios web (ver sección 2.5) aseguran una correcta interacción e intercambio de información entre ellas.
- El objetivo número 5 se centra en la capacidad del sistema de intercambiar datos de manera automática, es decir, sin intervención por parte del usuario. La adición de semántica y la propia definición del lenguaje se ha realizado para favorecer este objetivo. Mediante la acción conjunta de la ontología y el lenguaje, se puede hacer uso del conocimiento extraído de las aplicaciones y gracias a ellos es posible realizar una serie de operaciones que gestionan los intercambios de información de manera automática, resolviendo por sí solos las incompatibilidades que puedan surgir durante el proceso.

- El objetivo número 6 indica que se debe proporcionar una solución unificada a todos los niveles de la empresa. El lenguaje diseñado debe integrarse en una estructura mayor que permita la formación de un sistema completo que facilite la interacción con el usuario. Desde este punto de vista, gracias al lenguaje se permite la definición de diversas entidades que puedan participar en el sistema. De este modo, se pueden definir usuarios finales junto con sus propiedades, propietarios (tenants) de las aplicaciones y las propias aplicaciones con sus respectivos campos. De esta manera y conjuntamente con los entornos de computación en nube como soporte, se proporciona una solución unificada, útil para diferentes niveles de la escala empresarial.

A continuación se muestra la Tabla 6 que expone un cuadro resumen de la forma en la que la solución planteada abarca los objetivos propuestos, presentando una aproximación adecuada a los problemas que se pretenden resolver:

Nº Objetivo	Descripción Objetivo	Descripción Solución Propuesta
2	Definición de modelo de intercambio de información o lenguaje para representación del conocimiento.	Se ha diseñado una ontología capaz de capturar el conocimiento de las aplicaciones para que posteriormente el lenguaje pueda emplearlo en sus operaciones.
3	Dotar de semántica al lenguaje para la utilización del conocimiento y realización de inferencia y operaciones lógicas.	El lenguaje diseñado, CARL, empleará la semántica de OWL-DL, lo que permitirá la realización de operaciones de búsqueda e inferencia.
4	Facilitar interacción y transacciones entre aplicaciones en entornos de Cloud Computing.	La acción conjunta de CARL (gracias a sus especificaciones y operadores) y servicios web, habilita los intercambios de información entre aplicaciones.
5	Permitir intercambio de datos entre aplicaciones de manera automática.	Gracias a la semántica y al conocimiento extraído de las aplicaciones, es posible realizar operaciones que automaticen los procesos de intercambio de información.
6	Proporcionar una solución unificada en todos los aspectos de la empresa.	Las especificaciones del lenguaje permiten cubrir el abanico completo de niveles de la empresa y gracias a su utilización junto con los entornos de Cloud Computing, se consigue una solución unificada.

Tabla 6. Sumario soluciones de propuestas frente a objetivos planteados

Con toda la información comentada, se puede concluir que, tras estudiar diferentes opciones y alternativas, el lenguaje diseñado permite hacer frente a los objetivos propuestos al inicio de la investigación y que su utilización de manera conjunta con el resto de tecnologías estudiadas a lo largo del estado del arte, conforman una solución unificada y completa capaz de superar los retos planteados por la investigación.

En el siguiente capítulo se expondrán las diferentes decisiones de diseño tomadas en torno a los componentes adicionales necesarios para el funcionamiento global del sistema y el lenguaje.

4 Diseño de la solución

Este capítulo presenta los detalles de la solución planteada que permiten afrontar los problemas explicados con anterioridad y abordar las hipótesis expuestas. La solución propuesta pasa por el modelado de un lenguaje basado en ontologías mediante las cuales se captura el conocimiento de las aplicaciones existentes en una infraestructura de Cloud Computing y permiten el intercambio de datos entre ellas a través de servicios web. Sin embargo, el lenguaje y la ontología no pueden actuar por sí solos y precisan de un sistema en el que realizar sus actividades. A lo largo del capítulo se explicará el diseño completo de este sistema y toda su funcionalidad.

Tanto la arquitectura del sistema completo como la descripción de cada una de las partes que componen la solución proporcionada se detallan a continuación. Asimismo se explica el funcionamiento del lenguaje y todos los conceptos que participan en el proceso de intercambio de información.

4.1 Introducción

Para comprender el funcionamiento completo del lenguaje, éste debe situarse en el contexto en el que será utilizado. Por ello, esta sección comenta a grandes rasgos el funcionamiento global del sistema en el que el lenguaje aporta sus propiedades para facilitar la interoperabilidad. También se proporciona una primera aproximación de cada una de las entidades o roles que se encuentran presentes en el sistema y que son necesarias para el correcto funcionamiento del mismo.

CARL es un lenguaje semántico basado en ontologías que se encarga de capturar el conocimiento de las aplicaciones para, a posteriori, utilizar esa información de forma que se permita la comunicación entre aplicaciones heterogéneas. Dentro del sistema completo, la ontología diseñada (CARL Ontology) tomará un rol de ontología fundamental, es decir, la ontología que representará cada uno de los conceptos que podrán ser representados en cada una de las aplicaciones que formarán parte de la estructura de Cloud Computing y a partir de los cuales se representará todo lo demás.

Al conocer esto, surge una pregunta de forma inmediata, ¿Cómo recoge CARL la información de las aplicaciones en la nube? En todo sistema de Cloud Computing se hace necesaria una operación de despliegue de las aplicaciones para que la infraestructura pueda alojar cada aplicación, distribuirla y proporcionar todas las características del paradigma. En este caso, a la vez que se solicita la propia aplicación para ser desplegada en la nube, la persona encargada de esta tarea debe introducir una información referente a su aplicación que será entonces capturada y estructurada por CARL. Esta información consistirá en qué campos maneja la aplicación y el tipo de datos de cada uno de ellos y se podrá introducir a través de un

formulario habilitado para ello o adjuntando un fichero con extensión ".carl" como se ha detallado en el capítulo anterior (ver sección 3.3.3).

En la situación planteada surgen una serie de problemas. Ejemplos de ellos son la posible duplicidad de campos o que los conceptos de la ontología crezcan de forma descontrolada. Para evitar esto, se estudió la posibilidad de introducir una ontología del dominio que permitiese la representación del conocimiento de forma acotada. A pesar de que en un primer momento se pensó en diseñar esta ontología desde cero, tras un análisis previo se decidió emplear la ontología GoodRelations (Hepp, 2008), un vocabulario centrado en el dominio del comercio electrónico pero que presenta un diseño general que puede ser aplicado a otros dominios. Más información al respecto se proporciona en la sección 4.4.1.2 de este mismo capítulo.

Por último resulta imperativo introducir una entidad capaz de gestionar las diversas actividades necesarias para asegurar un correcto intercambio de información entre las aplicaciones. Esta entidad será un motor encargado de recibir la información enviada desde las aplicaciones y de ejecutar el conjunto de operaciones precisas para alcanzar la interoperabilidad de manera automática. Información más detallada sobre este componente se presenta en la sección 4.3.2.1, junto con otros elementos clave que forman parte de la arquitectura del sistema. La Figura 34 muestra un esquema conceptual de alto nivel donde se reflejan las entidades más relevantes que participan en el proceso completo:

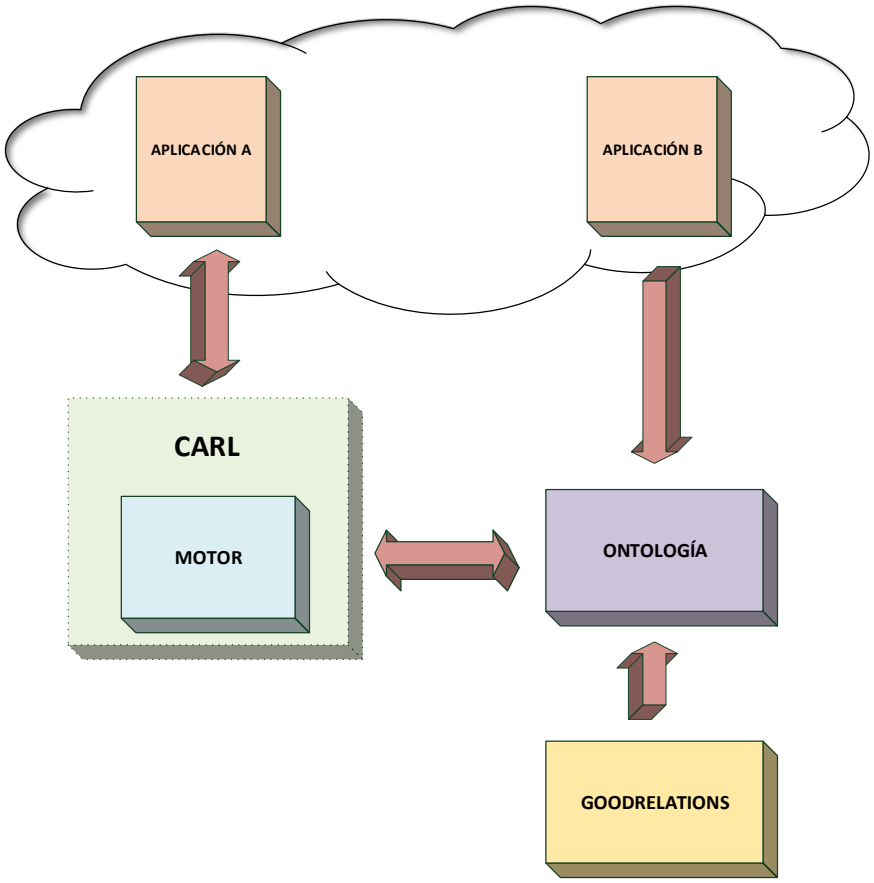


Figura 34. Esquema funcional del sistema

Siguiendo el esquema de la Figura 34 se puede observar como si dos aplicaciones quieren intercambiar datos, se establece una comunicación con el sistema (que será vía servicios web) solicitando los datos concretos. Este mensaje será procesado por el motor, el cual se encargará de gestionar el proceso completo con el resto de componentes participantes en el sistema y de solucionar los problemas de incompatibilidades de datos realizando las conversiones que se precisen en cada caso.

A lo largo de este capítulo se explican en detalle la arquitectura del sistema, cada una de las entidades participantes en el mismo, los motivos que han llevado a su utilización, su configuración y la funcionalidad completa que aportan. De esta manera se podrán comprender de una manera clara cada uno de los pasos seguidos para alcanzar la interoperabilidad en este tipo de entornos.

4.2 Concepto de "Jardín Vallado"

El concepto de "jardín vallado", que proviene del término "*walled garden*" en inglés, es una analogía utilizada en el ámbito de la Informática y las Comunicaciones para representar la situación de una plataforma informática o industria. En el campo de los medios y telecomunicaciones el concepto se encuentra relacionado con el control que ejerce una empresa desarrolladora o proveedora de servicios sobre las aplicaciones o el contenido del que se dispone en la plataforma y con la capacidad de la empresa de restringirlos (decisiones sobre qué aplicaciones son incluidas, cuáles estarán disponibles, requisitos que deben cumplir, etc.).

En general, un "jardín vallado" se refiere a la imposibilidad de los usuarios de moverse libremente en un entorno o de acceder a toda la información de un servicio. Justo desde esta perspectiva se explica la analogía del jardín, ya que la única manera de entrar y salir del jardín es mediante los puntos de entrada y salida diseñados para tal efecto.

Algunos ejemplos de utilización de este principio se encuentran en empresas tan extendidas e importantes como lo son Facebook o Apple, cuyo sistema operativo (iOS) hace uso de este mismo concepto.

Como se puede ver, este concepto se encuentra extendido entre las aplicaciones y sistemas de información actuales y es uno de los principales en los que se basa la investigación realizada. La pregunta inevitable llegado este punto es: ¿Por qué es tan importante limitar el entorno y aplicar este principio? La respuesta es que actualmente resulta imposible gestionar de manera única todos los sistemas e información existentes. Los trabajos previos que han intentado alcanzar la interoperabilidad a nivel global han fracasado estrepitosamente a pesar de los grandes objetivos marcados y de las inversiones realizadas para alcanzarlos. Es por ello que el enfoque de esta investigación se centra en la consecución de la interoperabilidad en entornos limitados (que no necesariamente tienen que ser más reducidos). En este trabajo se entienden como este tipo de entornos limitados aquellos en los que los usuarios que quieren participar en ellos deben aceptar una serie de prerrequisitos o reglas para poder hacerlo. Estos

requerimientos permiten acotar el alcance y proporcionar la información necesaria para que no se pierda funcionalidad alguna a la vez que se evita el problema de un mundo abierto en el que todo tiene cabida.

Como se justificaba en la sección 1.5.2, los entornos de Cloud Computing permiten establecer los límites de este jardín vallado a la vez que resulta posible aprovechar todo su potencial, convirtiéndose en el sustento principal de la investigación desarrollada.

Así, gracias al "jardín vallado" establecido, cuando un usuario desee que su sistema funcione en la estructura de computación en nube que engloba a CARL, deberá proporcionar suficiente información relativa a su aplicación como para que el sistema pueda capturar ese conocimiento a través de la ontología (campos presentes en su aplicación, tipo de datos de cada campo, etc.) y usarlo para realizar operaciones lógicas y automáticas que mejoren el rendimiento global de la plataforma y permitan la comunicación entre aplicaciones que, en principio, poseen propiedades y características completamente diferentes.

De esta manera, se evitan los problemas mencionados en cuanto a las incompatibilidades que existen en sistemas completamente abiertos donde resulta imposible establecer una correspondencia entre los datos tanto a nivel sintáctico como semántico. Mediante el funcionamiento conjunto del "jardín vallado", formado por la estructura de Cloud Computing empleada, y las tecnologías semánticas representadas en CARL y GoodRelations, se da solución a las posibles incompatibilidades, creando un entorno limitado en el que la comunicación entre aplicaciones heterogéneas es posible.

4.3 Diseño de la arquitectura

En este apartado se presenta la arquitectura general del sistema en el que se integra el lenguaje semántico diseñado. En ella se representan los diferentes componentes necesarios y las relaciones que se establecen entre ellos para asegurar el correcto funcionamiento del lenguaje y alcanzar los objetivos propuestos.

Para poder alcanzar el diseño final de la arquitectura, se ha realizado un análisis previo de alternativas para determinar el diseño arquitectónico que mejor se adapta a los requisitos planteados. A continuación se muestran las alternativas encontradas y posteriormente la discusión de las mismas que permitirá la elección de la más acertada.

4.3.1 Alternativas al diseño de la arquitectura

La toma de decisión del diseño de la arquitectura que debe adoptar el sistema no es un proceso trivial. Diversos factores deben tenerse en consideración para estar en disposición de elegir la arquitectura que mejor se adapte a la funcionalidad que se desea conseguir. En esta sección se dará solución a este problema y se seleccionará la opción más adecuada para alcanzar los objetivos de la investigación.

Una arquitectura software define el diseño de más alto nivel de la estructuración de un sistema e indica su funcionamiento, estructura y la interacción entre cada una de las partes de dicho software (Shaw & Garlan, 1996). Otra forma de definir este concepto, su importancia y alcance se proporciona en (Kruchten, Obbink, & Stafford, 2006) donde el autor afirma que "la arquitectura de software está relacionada con el diseño e implementación de estructuras software de alto nivel y es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de rendimiento de un sistema, así como otros no funcionales como pueden ser la fiabilidad, escalabilidad, portabilidad o la disponibilidad". Basándose en estas afirmaciones es sencillo comprobar cómo la elección de la arquitectura a desarrollar depende de un número elevado y variable de factores.

A continuación se presentan las opciones que se han tenido en cuenta en el proceso de decisión del diseño arquitectónico.

4.3.1.1 Arquitectura cliente-servidor

Esta arquitectura es un modelo de aplicación distribuida en el que las tareas se reparten entre proveedores de recursos (servidores) y los demandantes (clientes). Su funcionamiento consiste en que el cliente realiza peticiones a otro programa (servidor) que le proporciona una respuesta (Berson, 1996). Es posible emplear este sistema en programas ejecutados en una sola máquina, sin embargo sus propiedades son más ventajosas si se utilizan en sistemas multiusuario distribuidos, como es precisamente el caso del que se ocupa esta investigación.

En este tipo de arquitecturas la capacidad de proceso se encuentra repartida entre las figuras de los clientes y los servidores. Aún así, sus ventajas más relevantes son de tipo organizativo, ya que permiten una centralización en la gestión de la información y una separación en las responsabilidades que facilitan en gran medida el diseño del sistema y clarifica la estructura del mismo.

La arquitectura cliente servidor sustituye a la arquitectura monolítica utilizada ampliamente con anterioridad y que carece de distribución tanto a nivel físico como lógico (ambas presentes en las arquitecturas cliente-servidor) y que por lo tanto no se adaptaría a los requisitos expuestos en el sistema en cuestión.

Una disposición bastante habitual de las arquitecturas cliente-servidor son los sistemas multicapa (Bertocco, Ferraris, Offelli, & Parvis, 1998). En estos sistemas el servidor puede descomponerse en varios programas que pueden ejecutarse en diferentes computadoras, hecho que aumenta el grado de distribución del sistema.

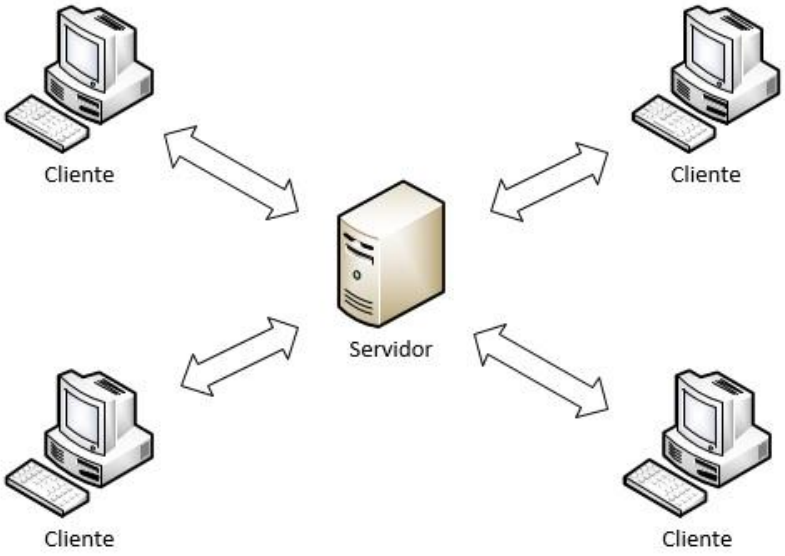


Figura 35. Estructura de arquitecturas cliente-servidor

Entre las características más interesantes de las arquitecturas cliente-servidor se encuentran las siguientes (Schuster, Jablonski, Kirsche, & Bussler, 1994):

- Cliente
    - Al ser el que inicia las peticiones, posee un papel activo en el proceso de comunicación.
    - Espera y recibe las respuestas del servidor.
    - Tiene la posibilidad de conectarse a varios servidores a la vez.
    - Suele interactuar de manera directa con usuarios finales a través de interfaces gráficas.

- Servidor
    - Desempeña un papel pasivo en el proceso comunicativo al esperar las solicitudes por parte de los clientes.
    - Al recibir una solicitud, se procesa y se envía la respuesta al cliente.
    - Aunque en ocasiones puede encontrarse limitado, pueden aceptar conexiones por parte de un gran número de clientes.
    - No interactúan con usuarios finales de manera habitual.

Las principales ventajas que proporcionan este tipo de modelos son:

- Centralización del control. Los accesos, recursos e integridad de los datos se encuentran controlados por el servidor de forma que se puedan evitar daños por parte de clientes defectuosos o no autorizados. Con esta característica también se facilitan las tareas de actualización.
- Escalabilidad. Cualquier elemento es susceptible de ser aumentado o mejorado en cualquier momento.

- Facilidad de mantenimiento. Las responsabilidades se encuentran distribuidas entre ordenadores independientes. Este hecho facilita la posibilidad de reemplazar, reparar, actualizar o trasladar servidores mientras que los clientes no se ven afectados por los cambios (encapsulación).
- Existen tecnologías diseñadas para este paradigma que aseguran la seguridad de las transacciones, la facilidad de empleo y la claridad de la interfaz de usuario.

Las desventajas más relevantes que se han encontrado en la construcción de este tipo de arquitecturas se muestran a continuación:

- Uno de los principales problemas del paradigma cliente-servidor ha sido la congestión de tráfico. Si existe un gran número de peticiones simultáneas a un mismo servidor, es posible que se le causen muchos problemas.
- Cuando un servidor se cae, las peticiones de sus clientes pueden no ser satisfechas.
- El software y hardware son muy determinantes. Normalmente se precisa de software y hardware específico, especialmente en el lado del servidor, para satisfacer adecuadamente el trabajo, hecho que conlleva un aumento de costes.
- El cliente no dispone de los recursos que puedan existir en el servidor.

Con posterioridad a la aparición de las arquitecturas cliente-servidor, comenzaron a desarrollarse las arquitecturas multicapas. Desde ese momento una arquitectura cliente-servidor puede considerarse una arquitectura genérica de dos capas. Este tipo de arquitecturas multicapas será estudiado en mayor detalle en la siguiente sección.

4.3.1.2 Arquitectura dividida en capas

Las arquitecturas divididas en capas surgen de la necesidad de establecer una separación entre la lógica de negocio y la lógica del diseño (Czarnecki & Eisenecker, 2000). De ahí que se consideren una “evolución” de las arquitecturas cliente-servidor.

Una de sus principales ventajas es que si es necesario realizar modificaciones sobre el sistema, únicamente es necesario trabajar sobre el nivel específico, evitando revisiones innecesarias que incurren en aumentos de coste y tiempo. Además de esto, con esta arquitectura se hace posible una distribución del trabajo por niveles que permiten a diferentes que grupos de trabajo puedan trabajar de manera independiente, abstrayéndose del resto de niveles y conociendo únicamente la API que exista entre cada uno de ellos.

Actualmente, uno de los diseños de arquitecturas más empleados son las arquitecturas multinivel. En ellas se responsabiliza a cada nivel de una misión sencilla lo que permite que se construyan arquitecturas de gran escalabilidad, aumentando sus capacidades fácilmente para adaptarse a las necesidades del entorno. Dentro de las arquitecturas multinivel las más ampliamente extendidas son las de tres niveles (capas).

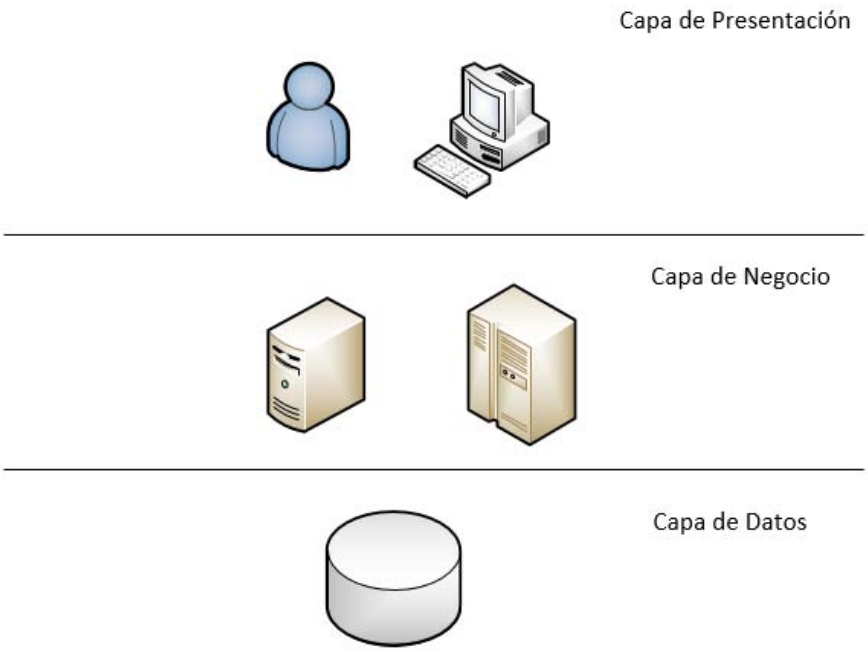


Figura 36. Arquitectura clásica dividida en tres capas

La Figura 36 muestra la división habitual que se establece en las arquitecturas de tres capas. A continuación se proporciona una descripción más detallada de cada uno de los niveles en que se divide:

- Capa de Presentación. Se encuentra en contacto con el usuario final. Comunica la información al usuario y captura también la información proporcionada por el mismo. Esta capa es conocida habitualmente como la interfaz gráfica y debe resultar sencilla y comprensible para el usuario. Esta capa solamente se comunica con la capa de negocio.
- Capa de Negocio. Es el nivel en el que residen los programas que se ejecutan. Reciben peticiones del usuario y se envían las correspondientes respuestas tras la realización de los correspondientes procesos de la lógica de negocio. Esta capa establece comunicación con la capa de presentación con la que intercambia solicitudes y resultados, y con la capa de datos, de manera que se pueda solicitar al gestor de la información almacenar o recuperar datos de él. También se consideran dentro de este nivel los programas de aplicación.
- Capa de Datos. Es el lugar donde residen propiamente los datos y el nivel que se encarga de acceder a los mismos. Se encuentra formada por uno o más gestores de bases de datos que realizan el proceso completo de almacenamiento, recibiendo solicitudes de recuperación o almacenaje de información desde la capa superior (capa de negocio).

Con esta división en niveles se puede observar cómo se encuentran por un lado los clientes que interactúan con los usuarios finales, por otro los servidores de aplicación y la lógica que permite procesar la información para los clientes y por último los servidores de bases de datos que los almacenan y facilitan a los servidores de aplicación. Cada nivel posee una misión claramente definida que ayuda a una mayor claridad conceptual del diseño.

Algunas de las ventajas que proporcionan estos modelos son:

- Abstracción
- Aislamiento
- Escalabilidad
- Mejoras en pruebas
- Independencia

Sin duda, la arquitectura dividida en capas se presenta como una opción interesante y factible que, en el caso del que se ocupa este documento, podría resultar especialmente beneficiado de las posibilidades y características que ofrecen las plataformas de Cloud Computing, distribuyendo los contenidos y configurando las comunicaciones para sacar el máximo partido de la separación conceptual y arquitectónica del sistema.

4.3.1.3 Modelo Vista Controlador (MVC)

El Modelo Vista Controlador (MVC) es un patrón de arquitectura software que establece una clara separación entre los datos y la lógica de negocio de una aplicación de su interfaz de usuario y el módulo gestor de eventos y comunicaciones (Burbeck, 1987). Para ello, el MVC propone la construcción de tres componentes: el modelo, la vista y el controlador. De esta manera se definen por un lado los componentes para la representación de la información y por otro la interacción del usuario.

Un aspecto que destaca en este patrón de diseño es que se encuentra basado en las ideas de reutilización de código y separación de conceptos, propiedades cuyo objetivo es facilitar las tareas de desarrollo y mantenimiento, aspectos sin duda muy relevantes para sistemas de grandes dimensiones especialmente.

Genéricamente, los elementos del MVC pueden definirse de la siguiente manera (Leff & Rayfield, 2001):

- El Modelo. Representa la información con la que opera el sistema, es decir, actúa como gestor de acceso a la información. También implementa privilegios de acceso que hayan sido descritos en las especificaciones de la aplicación (lógica de negocio). Se encarga de enviar a la Vista la información que se solicite y deba ser mostrada en cada momento. Las peticiones tanto de acceso como de manipulación de la información contenida llegan al Modelo a través del Controlador.

- El Controlador. Da respuesta a eventos, generalmente provocados por el usuario, y envía peticiones al Modelo cuando se precise la información que contiene. También puede enviar comandos a la Vista asociada al solicitar cambios en la forma en la que se debe presentar el Modelo. De todo esto se extrae que el Controlador actúa de intermediario entre la Vista y el Modelo.
- La Vista. Presenta el Modelo (datos y lógica de negocio) en el formato apropiado en cada caso. Por este motivo precisa de la información contenida en el Modelo para poder representarla en la salida.

La Figura 37 muestra la estructura que forman cada uno de los componentes y las relaciones que se establecen entre ellos.

Es importante destacar que tanto la Vista como el Controlador dependen del modelo aunque el modelo no depende de ninguno de los dos. Este es uno de los beneficios principales de la separación. Esta separación permite que el modelo sea construido y probado independientemente de su representación visual. La separación entre la Vista y el Controlador es secundaria en muchas aplicaciones y, de hecho, muchos marcos de interfaces de usuario implementan este rol como un objeto. Por otro lado, en aplicaciones web la separación entre la Vista (navegador) y el Controlador se encuentra muy bien definida.

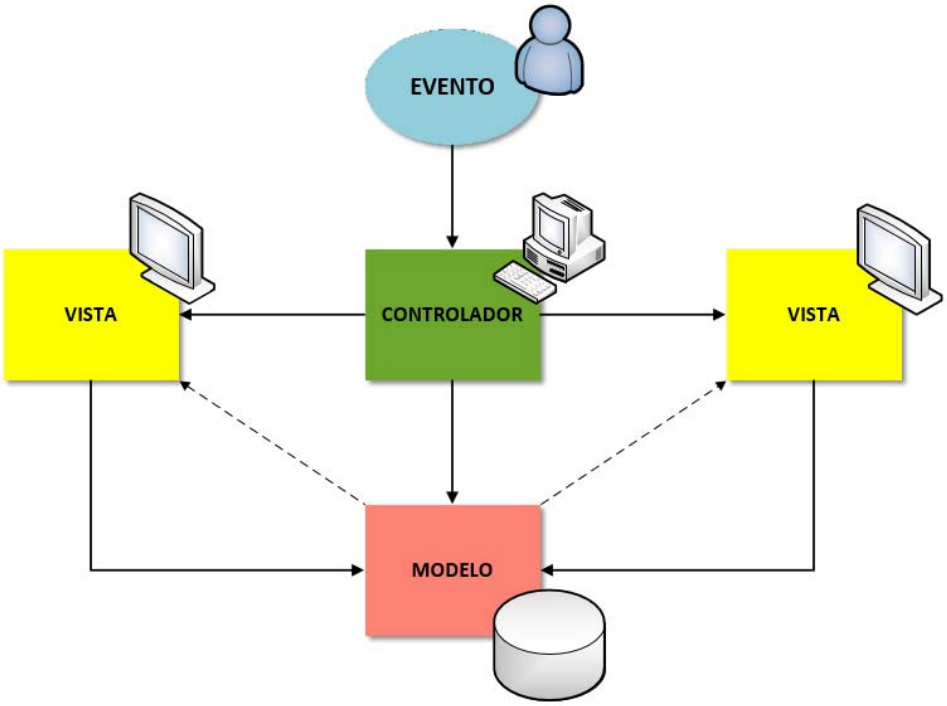


Figura 37. Estructura del patrón Modelo Vista Controlador

Debido a la popularidad adquirida por el patrón, se pueden encontrar diferentes variaciones e implementaciones del MVC. A pesar de esto, el flujo de control que suele seguirse es generalmente el mismo:

1. El usuario interactúa con la interfaz (Vista)
2. El Controlador recibe (a través de la Vista) la notificación correspondiente al evento solicitado por el usuario y lo gestiona habitualmente a través de un gestor de eventos.
3. El Controlador accede al Modelo y lo actualiza. Este proceso se realiza en concordancia con la acción solicitada por el usuario en un primer momento.
4. El Controlador delega entonces a los objetos de la vista la tarea de desplegar la interfaz de usuario. La Vista genera la interfaz obteniendo los datos del Modelo donde ya se reflejan los cambios solicitados. El Modelo no debe poseer conocimiento directo de la Vista.
5. Una vez finalizado este proceso, la interfaz de usuario queda a la espera de nuevas interacciones por parte del usuario que originen un nuevo comienzo del ciclo aquí detallado.

La Figura 38 muestra la colaboración que se establece entre cada uno de los componentes que forman el MVC:

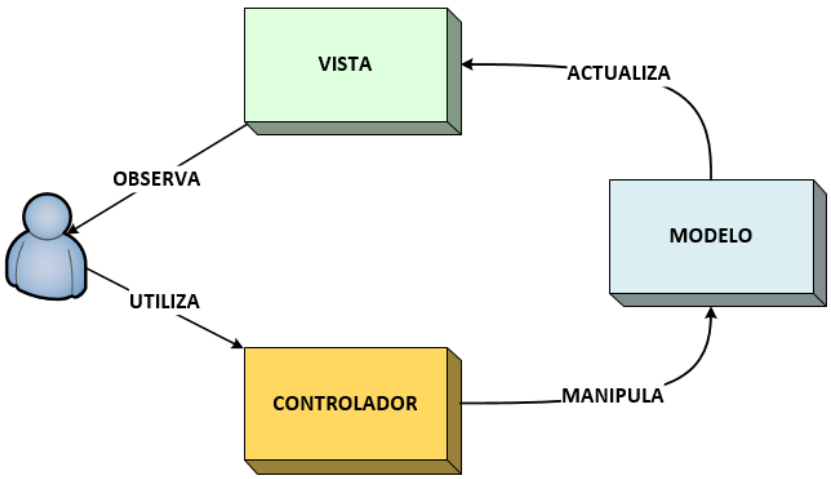


Figura 38. Colaboración entre los componentes del MVC

Por último es importante comentar que muchos sistemas de información emplean sistemas gestores de bases de datos parta manejar la información que debe utilizar la aplicación (ReensKaug, 2003). En el marco del patrón MVC, el Modelo sería el encargado de dicha gestión. Por lo tanto, la unión entre las capas de presentación y negocio previamente explicado en el paradigma de las arquitecturas divididas en capas (sección 4.3.1.2), representaría la integración entre la Vista y el Controlador correspondiente. De esta manera el MVC no desea discriminar entre su capa de negocio y la capa de presentación, sin embargo quiere separar la parte gráfica visual de su lógica y datos, hecho que permite una mejora sustancial en el desarrollo y mantenimiento tanto de la Vista como del Controlador de forma paralela, debido a que ambos realizan ciclos de vida que difieren mucho entre sí.

4.3.1.4 Discusión

Una vez planteadas las principales opciones seleccionadas, se hace necesaria una discusión que permita alcanzar una conclusión acerca de qué diseño arquitectónico es el más adecuado para estructurar el sistema.

Comprobando las características de todos los diseños resulta sencillo concluir que el sistema objetivo podría ajustarse a cualquiera de los diseños estudiados, sin embargo, las propiedades particulares de cada uno de ellos establecen ciertas variaciones que pueden resultar más o menos adecuadas para los objetivos de este trabajo.

En primer lugar, la arquitectura cliente-servidor cubre el espectro de requisitos existentes y presenta un diseño sencillo que se adapta adecuadamente a las propiedades de los sistemas de Cloud Computing dada su naturaleza distribuida y su funcionamiento basado en la solicitud de información desde un cliente a un servidor. Aún así, este diseño resulta demasiado simple como para representar el gran número de actividades y procesos que se tienen que cubrir a la hora de comunicar unas aplicaciones con otras. Adicionalmente, esta arquitectura puede plantear problemas debido a la gran concurrencia que se puede dar al ser posible que muchas aplicaciones actúen como clientes y servidores simultáneamente pudiendo provocar problemas en el tráfico. Por último y no menos importante, existe el problema de la ausencia de una división lógica o conceptual del sistema que, además de resultar más complicado de entender, provoca que las tareas de modificación y mantenimiento sean mucho más complejas, que en última instancia hacen que esta opción sea descartada debido al alto grado de variabilidad y personalización que intrínsecamente conllevan este tipo de sistemas de computación en nube.

Los dos siguientes modelos o paradigmas arquitectónicos se diferencian del anterior en que establecen una división entre la lógica de negocio y los datos, aspecto fundamental en una gran mayoría de los sistemas de información. Ambos separan, aunque de formas algo distintas, los aspectos de presentación al usuario, la lógica de negocio y los datos necesarios para su funcionamiento.

Ambas opciones parecen factibles a la hora de representar el sistema. Desde muchas perspectivas pueden resultar incluso modelos paralelos que permiten estructurar un mismo entorno de manera semejante. Sin embargo, existen algunas diferencias entre ambos que pueden inclinar la balanza hacia uno de ellos.

En primer lugar encontramos las dependencias existentes dentro de los propios modelos. El MVC es un patrón de diseño y no en sí una arquitectura por definición. Podría considerarse una personalización algo flexible del patrón dividido en capas. El patrón por capas indica que las capas superiores (más dependientes de cada aplicación) tienen dependencias hacia abajo, pero las capas de abajo (más generales) no tienen dependencias hacia arriba. De esta manera se provoca que las capas de arriba hacia abajo puedan ser intercambiadas con un menor esfuerzo al no estar acopladas entre sí. En el patrón MVC por otro lado, implica que el Controlador recibe los eventos de la vista y existe una clara dependencia entre ambos, produciendo un fuerte acoplamiento entre ellos.

Otra diferencia reside en que en el MVC las comunicaciones no se encuentran separadas, sino integradas en el modelo junto con los datos, lo que a nivel conceptual y dado el caso que nos ocupa puede resultar algo confuso. En una arquitectura tradicional dividida en capas las comunicaciones se presentan como un aspecto más flexible y en función de su naturaleza su gestión se puede adaptar más fácilmente. En este caso en el que el trabajo se desarrolla en entornos de computación en nube, parece más adecuado que estos procesos posean mayor libertad y los servicios web encargados de la comunicación entre las aplicaciones alojadas en la infraestructura puedan gestionarse de manera personalizada para un mejor manejo del conocimiento contenido en cada una de las aplicaciones y que deberá ser capturado por el sistema global.

Por estas razones la elección tomada es la de diseñar una arquitectura basada en el modelo dividido en capas (particularmente tres) que serán las encargadas de estructurar conceptual y arquitectónicamente todos los contenidos existentes en el sistema y que permitirán un diseño adecuado para alcanzar los objetivos propuestos en la investigación.

A lo largo de las siguientes secciones se proporciona una descripción detallada de los aspectos de diseño más relevantes tanto a nivel de la arquitectura general del sistema como de la arquitectura diseñada para el lenguaje semántico CARL.

4.3.2 Diseño final de la arquitectura

Tras el proceso de decisión comentado previamente, se ha optado por emplear una arquitectura dividida en tres capas que permitirá establecer una división conceptual y funcional acorde con los objetivos del sistema. Con el diseño de esta arquitectura se detallan cada uno de los componentes que trabajarán conjuntamente con CARL para gestionar las aplicaciones y sus procesos de comunicación.

A continuación se presenta el diseño final de la arquitectura del sistema y se detallan cada uno de los módulos y funcionalidades de los que se componen.

4.3.2.1 Arquitectura general del sistema

El diseño elegido para guiar el funcionamiento del lenguaje en el sistema de computación en nube es el de una arquitectura dividida en capas. La ventaja de utilizar este tipo de arquitecturas es la distinción conceptual y funcional entre cada una de las capas. Cada capa posee una funcionalidad precisa y particular y entre ellas se establece una relación de dependencia de las capas inferiores hacia las superiores, lo que significa que las capas superiores dependen de las funcionalidades proporcionadas por las capas inferiores.

Las arquitecturas por capas se utilizan en numerosos dominios como por ejemplo los que se reflejan en (Davulcu, Freire, Kifer, & Ramakrishnan, 1999), en (Simmons, Goodwin, Haigh, Koenig, & O'Sullivan, 1997) o (Engel et al., 2003), pero son especialmente relevantes en sistemas de comunicación donde cada capa implementa un aspecto diferente del intercambio de información. Por ejemplo, la capa más profunda habitualmente representa la parte más básica de la comunicación en un nivel físico, es decir, las señales físicas eléctricas de la comunicación.

En la arquitectura del sistema CARL la funcionalidad se divide en la capa de presentación o Interfaz Gráfica de Usuario (GUI) del PaaS, una segunda capa donde se encuentra la Lógica de Negocio y por último el nivel de almacenamiento y persistencia de datos. Cada nivel posee una funcionalidad diferente que permite afrontar los retos que plantea la interoperabilidad de aplicaciones complejas.

La arquitectura presentada es una solución tecnológica diseñada a medida para resolver los problemas reflejados en esta investigación y ha servido como base conceptual para la implementación del sistema utilizado en la fase de validación de las hipótesis de esta tesis doctoral.

A continuación, en la Figura 39, se detallan los diferentes componentes que forman parte de la arquitectura y que resultan necesarios para el correcto funcionamiento global del sistema:

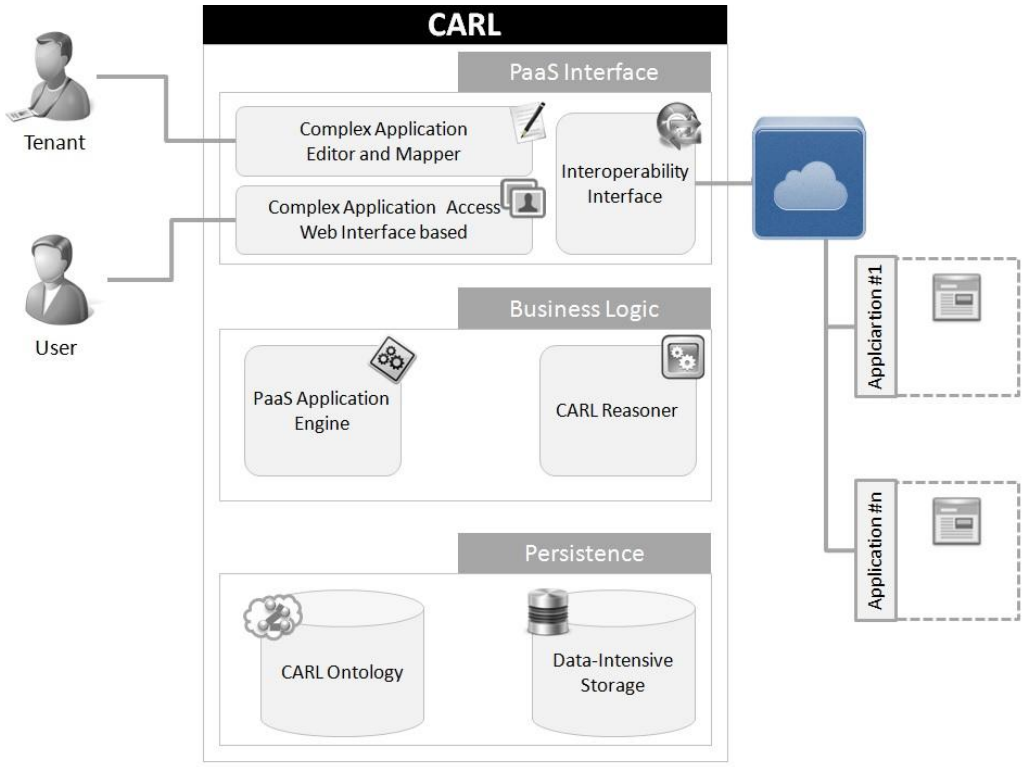


Figura 39. Arquitectura de CARL

Cada uno de los módulos presentados realiza una función clara y definida que contribuye a alcanzar los resultados deseados. Seguidamente se explica en detalle cada uno de los componentes participantes en el sistema:

- **Complex Application Editor and Mapper**. Este componente interactúa con los usuarios expertos o tenants, proporcionando un conjunto de elementos gráficos para anotar los recursos semánticamente basados en la ontología CARL y en la ontología del dominio seleccionada (GoodRelations). Además de los elementos gráficos para anotar las propiedades se proporcionan herramientas de administrador para gestionar las aplicaciones de las que son propietarios.
- **Complex Application Access Web Interface-based**. Este componente ofrece interacción sencilla con las aplicaciones que se encuentran en el PaaS. Es la visión del sistema que tienen los usuarios finales que se suscriben a las aplicaciones existentes en la estructura de computación en nube. Este módulo proporciona también un sistema basado en recuperación de información, fundamentado en el razonador de CARL, para localizar las aplicaciones existentes. Este buscador realiza operaciones de inferencia para ofrecer al usuario diversas alternativas relacionadas con la búsqueda realizada.
- **Interoperability Interface.** Este componente se encarga de la gestión de la comunicación entre las aplicaciones y el sistema de forma que se pueda completar el proceso de interoperabilidad. A su vez, este módulo se puede descomponer en dos niveles diferentes y necesarios que se presentan en la Figura 40:

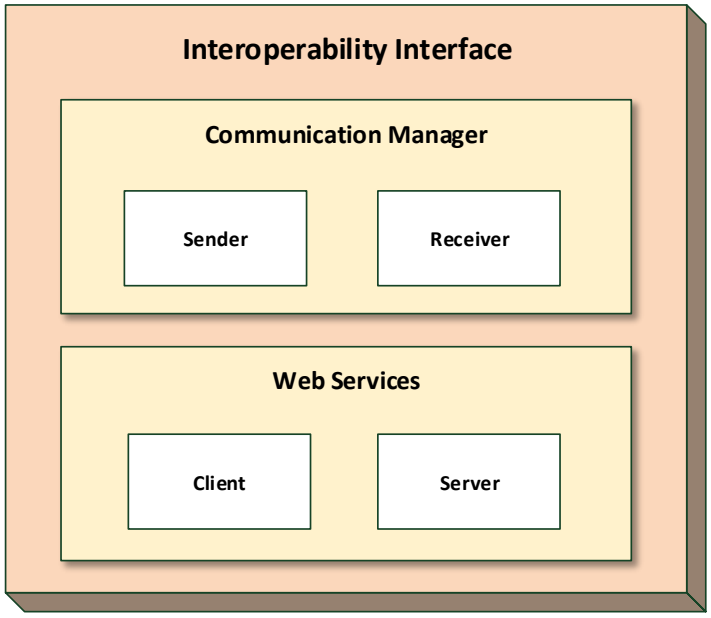


Figura 40. Descomposición de Interoperability Interface

Estos dos niveles presentados se encargan de la funcionalidad básica a la hora de establecer procesos de comunicación entre las aplicaciones y el sistema. A continuación se detallan cada uno de ellos en mayor profundidad:

1. *Web Services*. Este nivel habilita el nivel más bajo de comunicación. El propósito del nivel de servicios web es el de proporcionar acceso a dicha tecnología, necesaria para que se produzca la comunicación, ocultando los detalles de implementación de esos servicios web. Consiste en un conjunto de servicios web simples que se encuentra dividido en dos componentes principales: un servicio web cliente y otro servidor. Estos componentes interactúan con los niveles más bajos de la comunicación y transporte a nivel de la pila de red.

2. *Communication Manager.* Su objetivo es el de abstraer todo el proceso de comunicación entre los servicios web y el motor. Permite realizar dos funciones: en primer lugar, proporciona la información recibida desde otras entidades y por último, se le provee de aquello que debe ser enviado a la otra entidad participante en el proceso de comunicación. Está formado por dos componentes principales denominados *sender* (emisor) y *receiver* (receptor):

    - El emisor emplea servicios web para enviar la información que se precise.
    - El receptor se encarga de la recepción de la información proveniente de las entidades que participan del proceso de comunicación.

La información que llega a la Interoperability Interface y que parte de ella a través de los servicios web se encuentra en forma de WSDL. A partir de esta descripción del servicio se inicia el proceso de interoperabilidad que se inicia en el PaaS Application Engine.

- **PaaS Application Engine**. Es el núcleo de la semántica del PaaS. Su funcionalidad se basa en la gestión de los procesos de comunicación, gracias a la información recibida desde el Interoperability Interface, y en la conversión de datos mediante la interacción entre las peticiones provenientes de los servicios web y el razonador. Una explicación más detallada de este componente clave se encuentra en la sección 4.3.2.2 de este capítulo.
- **CARL Reasoner**. Es un componente básico para el funcionamiento del sistema. Utiliza procesos para encontrar conjuntos y subconjuntos de anotaciones basados en restricciones lógicas. Está basado en una implementación particular de los motores de razonamiento de OWL-DL, como el Renamed A-Box and Concept Expression Reasoner (RACER). El razonador trabaja extrayendo la información relevante de la ontología, participando activamente en el proceso de interoperabilidad. Posee tres funciones diferentes:

- Realiza procesos de inferencia contra la ontología cuando los usuarios hacen uso del buscador de aplicaciones. Estas operaciones consisten en ofrecer a los usuarios aplicaciones relacionadas con aquéllas que están buscando. De esta manera pueden descubrir otras aplicaciones interesantes o alternativas y que les pueden resultar de utilidad.
- Interactúa con el módulo el analizador semántico, localizado en el motor. El objetivo de esta interacción es proporcionar información relativa a la equivalencia entre la información recibida por las aplicaciones cuando realizan las peticiones a través de los servicios web y el conocimiento sobre los campos contenido en la ontología. Es decir, realiza consultas SPARQL para localizar las equivalencias de los campos involucrados en el proceso con aquellos a los que se encuentran unidos en la ontología del dominio.
- Un vez el motor conoce los campos precisos que debe localizar, se lo comunica al razonador que realiza nuevamente un consulta SPARQL contra la ontología para obtener la información requerida y devolverla al motor para que éste aplique la función de transformación correspondiente, en caso de ser necesario.

Como se ha comentado, el razonador está basado en motores de razonamiento de OWL-DL. Esto significa que es necesario que la información que llegue al razonador llegue en un formato legible para él. El encargado de realizar esta tarea es el intérprete, un módulo existente en el motor que se encarga de traducir la información a un formato entendible para el razonador (OWL). Sin la existencia de este componente sería imposible interactuar con la ontología a través del razonador y, por tanto, extraer la información relevante en cada momento para asegurar el correcto funcionamiento del sistema. Toda interacción que se produce en el sistema entre el PaaS Application Engine y el CARL Reasoner se sucede a través de la actuación del intérprete. Más información acerca de este componente se puede encontrar en la sección 4.3.2.2 de este mismo capítulo.

- **CARL Ontology**. Este componente software forma el sistema de almacenamiento semántico de datos que posibilita la persistencia de la ontología, las consultas realizadas por los componentes de la capa de la Lógica de Negocio y ofrece una capa de abstracción que permite el rápido almacenamiento y recuperación de grandes cantidades de información.
- **Data-Intensive Storage**. Este componente añade almacenamiento online donde los datos quedan almacenados en el IaaS proporcionado por la infraestructura de computación en nube subyacente.

Adicionalmente a los diferentes componentes que constituyen la arquitectura, existen dos figuras que interactúan con el sistema y sin los cuales éste no tendría sentido: tenants y usuarios. A continuación se describe la función de cada uno de ellos y sus posibilidades de interacción con el sistema:

- **Tenants**. Son los propietarios de las aplicaciones que se suben a la plataforma de Cloud Computing. Su misión consiste en registrarse en el sistema como tenants y proporcionar las aplicaciones, información sobre ellas, etiquetado de contenidos y el resto de aspectos que necesite cada aplicación particular para su funcionamiento a través del "Complex Application Editor and Mapper" de la arquitectura. Además, este tipo de usuario poseerá privilegios de administrador sobre sus aplicaciones, de forma que pueda gestionarlas y realizar las modificaciones que considere necesarias en cada momento.

- **Usuarios**. Esta figura representa a los usuarios finales de las aplicaciones alojadas en la plataforma de computación en nube. Para poder hacer uso de ellas, el primer paso que deben cumplir es el de registrarse. Deberán proporcionar los datos personales habituales así como cierta información económica que permita facturar los servicios a los que acceda en la plataforma.

  Una vez formalizado el registro, el usuario podrá realizar búsquedas en el sistema en función del tipo de aplicaciones en las que esté interesado a través del "Complex Application Access Web Interface-based". Esto es posible gracias a las etiquetas e información introducida por los tenants al subir sus aplicaciones a la plataforma. Cuando el usuario localice una aplicación de su interés, éste podrá suscribirse a ella, pasando a ser un usuario activo de la misma y siendo facturado por ello. Como es lógico, los usuarios podrán también darse de baja de las aplicaciones en cualquier momento.

  El valor añadido del sistema, la interoperabilidad, se produce cuando el usuario se ha suscrito a varias aplicaciones que le son de utilidad. Estas aplicaciones podrán comenzar a intercambiar datos de forma automática cuando lo requieran. Para permitir este tráfico de información el usuario debe aceptar unos términos y condiciones durante el registro que permita a las aplicaciones realizar estos procesos sin solicitar su consentimiento específico en cada ocasión.

Como se puede comprobar, todos los elementos que forman parte de la arquitectura tienen una función clara, definida y necesaria, pues la ausencia de cualquiera de ellas evitaría un correcto funcionamiento del sistema. Es en la capa de la lógica de negocio donde se encuentran los aspectos principales, puesto que todos los componentes que utilizan la información capturada por CARL actúan ahí. Aquí se analizan los mensajes que llegan a través de las comunicaciones mediante servicios web y se proporcionan soluciones mediante las acciones del motor y el razonador. Por tanto, la acción conjunta de todos estos componentes forma los fundamentos del sistema y permiten el establecimiento de la comunicación entre las aplicaciones existentes a través del lenguaje.

Como se ha comentado anteriormente, en la próxima sección se proporcionará información más completa acerca del funcionamiento del PaaS Application Engine o motor. Se ha decidido tratar este componente de forma aislada por el gran peso que posee en el funcionamiento global del sistema y para explicar de forma más detallada sus funciones y elementos, aumentando así el grado de entendimiento de sus actividades.

4.3.2.2 PaaS Application Engine (Motor)

El motor se puede definir como el componente principal del sistema. Se encarga de la gestión de los procesos de comunicación y de organizar la secuencia de acciones a realizar. Recibe la información de las aplicaciones a través de los servicios web provenientes de la Interoperability Interface, descompone estos mensajes, que llegan en forma de fichero WSDL, los analiza y actúa en consecuencia a la petición concreta de cada aplicación.

Para poder desarrollar estas actividades, el motor se descompone en una serie de módulos con funciones bien definidas. Estas funciones consisten en interpretar la información recibida por parte de los servicios web, trabajar con la ontología a través del razonador para establecer correlaciones entre las peticiones realizadas y la información que se debe proporcionar y gestionar las interacciones que se produzcan con el propio razonador. Cada uno de los módulos recibirá una entrada y producirá una salida que servirá a su vez de entrada al siguiente, obteniendo finalmente como resultado la acción que se debe realizar en cada momento. Esta descomposición del razonador en módulos se refleja gráficamente en la Figura 41:

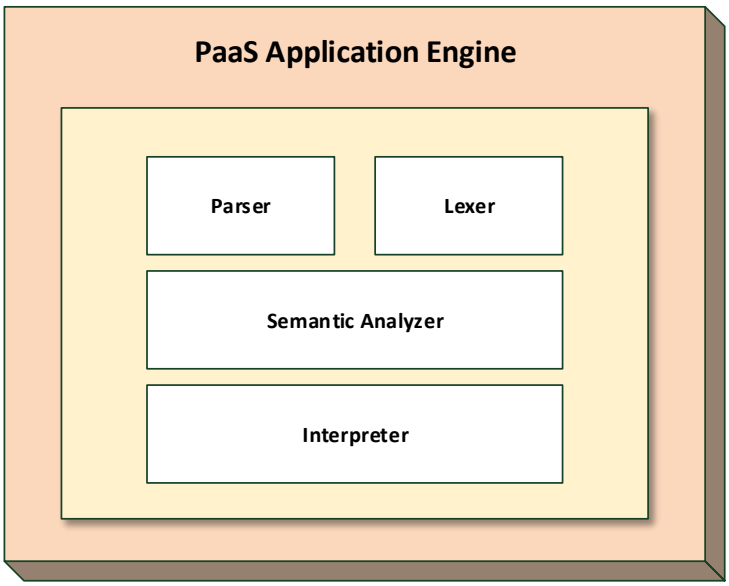


Figura 41. Descomposición del motor de CARL

A continuación se presenta una explicación detallada de la función a desempeñar por cada uno de los módulos:

- *Lexer o analizador léxico*. Se encarga de leer las entradas procedentes de la información enviada por las aplicaciones a través de los servicios web, ponerla en memoria y de devolver los tokens uno por uno.

- *Parser*. Este módulo reconoce la información relevante del mensaje mediante el análisis sintáctico de la estructura del conjunto de tokens extraídos del analizador

léxico. Esta operación se realiza por medio de los inputs y outputs que vienen indicados en el fichero WSDL.

- *Semantic Analyzer*. Recibe la información relevante obtenida a través del parser e interactúa con el razonador para que extraiga el conocimiento asociado a esa información a través de la ontología. Esta interacción entre el analizador semántico y el razonador se realiza previa intervención del intérprete, que transforma la información recibida a formato SPARQL para que sea comprensible para éste último. Una vez recibida la información relevante, la operación consiste en obtener la relación existente entre los campos demandados por la aplicación y la información proporcionada para su localización con los existentes en la ontología del dominio. Esto permite establecer un nexo de unión para localizar la información deseada. La salida proporcionada es, por tanto, un conjunto de sentencias de comparación de los campos enviados por la aplicación con sus equivalentes en la ontología del dominio en formato CARL construido por el analizador.

- *Interpreter*. Su cometido es el de realizar traducciones de la información que recibe a un formato comprensible para el razonador. Su actuación se precisa en dos momentos concretos del proceso de intercambio de datos:

    - Traduce texto a SPARQL para habilitar la comunicación entre el analizador semántico y el razonador.

    - Traduce sentencias CARL a OWL tras recoge el conocimiento procesado por el analizador semántico. El intérprete las traduce y transforma para proporcionar al razonador una entrada adecuada que permita a posteriori extraer la información requerida por la aplicación que originó el proceso de intercambio de información. Para la realización de esta tarea, el intérprete se basa en la información recogida en la Tabla 7, donde se recogen las equivalencias entre CARL y OWL-DL.

Como se puede comprobar, gracias a esta estructura se pueden capturar e interpretar las acciones a realizar en el sistema ya que se reconocen sentencias válidas del lenguaje, se procesan y se interpretan.

Para reflejar esta secuencia de acciones de forma más clara se presenta a continuación un ejemplo del funcionamiento de los componentes del motor:

- La aplicación de Telefónica desea conocer el número de cuenta de un cliente para realizar un cargo. Para ello, envía un servicio web solicitando el número de cuenta del cliente con DNI 12345678 el cual llega a la *Interoperability Interface*, donde es redirigido hacia el motor. En este momento se inicia la siguiente serie de pasos:

    1. El analizador léxico pone en memoria las líneas recibidas en el WSDL y las separa en tokens.

2. El parser recibe los tokens y selecciona los que resultan de interés mediante el estudio de los inputs y outputs reflejados en el WSDL. En el caso de este ejemplo se quedará con Telefónica, DNI, 12345678, NumeroCuenta, datos que pasará al analizador semántico.

3. El analizador semántico recibe los tokens relevantes, se comunica con el intérprete, que los traduce y los envía al razonador para que consulte en la ontología con qué campos de la ontología del dominio se corresponden los campos DNI y NumeroCuenta de Telefónica. En ese momento, el razonador devuelve los campos de la ontología del dominio con los que se alinean los campos requeridos y el analizador semántico reúne la información y lo expresa en formato CARL. Por ello, lo que el analizador semántico envía al intérprete es que DNI match(Individual:SerialNumber) y NumeroCuenta match(Offering:SerialNumber).

4. Esta información llega al intérprete, quien traduce las sentencias CARL para que sea comprensible por el razonador de OWL-DL.

5. A partir de aquí el razonador realizaría una consulta SPARQL que devolviese el valor de *Offering:SerialNumber* asociado al *DNI 12345678.*

Como opción adicional, la aplicación que solicita el dato puede comunicar a través del servicio web inicial la aplicación concreta en la que se debe buscar la información requerida, incluyendo esa información como un parámetro más de la consulta a la ontología. En caso de no indicarse la aplicación en la que buscar, se procederá como en el ejemplo y la consulta se realizará sobre la ontología completa.

Una situación alternativa que puede producirse durante la acción del motor, más concretamente en el paso final, es que el dato enviado por la aplicación que envía la petición y que sirve como información para localizar el campo requerido (en el caso del ejemplo anterior DNI), posea un tipo de datos diferente en la aplicación destino. El problema que surge es que al pasar la información al razonador para que éste ejecute la consulta que proporcione el dato deseado, la salida sea nula. El motivo es que se puede estar buscando el valor de DNI como un número entero y en las aplicaciones existentes en la ontología que poseen esa información esté representado como una cadena de caracteres. Ante esta situación, el motor actúa aplicando las funciones de conversión al dato de localización hasta que la consulta devuelva un resultado o se agoten las funciones de transformación compatibles. Si se diese este último caso, el sistema informa de que es imposible obtener el dato deseado pues o no se encuentra almacenado en ninguna aplicación, o posee un formato incompatible con el de la aplicación que lo solicita. Si mediante alguna de estas transformaciones se consigue el dato, el proceso continúa normalmente.

Como se comentaba, una vez que se han analizado los mensajes provenientes de las aplicaciones, el motor envía al razonador la información necesaria para que éste obtenga los datos que necesita. Una vez el razonador consiga esos datos, vuelven a enviarse al motor para que ejecute la función de transformación necesaria para solucionar las incompatibilidades. Sin embargo, cabe recordar que el razonador empleado es un razonador de OWL-DL y para que se comprendan las órdenes enviadas en sintaxis de CARL debe existir un paso previo en el que se establezcan una serie de equivalencias entre la sintaxis presentada por CARL y OWL-DL. El

encargado de realizar esta traducción, como ya se ha explicado, será el intérprete y para estar en disposición de hacerlo empleará la información contenida en la Tabla 7 que se muestra a continuación:

CARL	OWL-DL
Define	Class
subConceptOf	subClassOf
Field	Property
Match	EquivalentProperty
Type	DatatypeProperty
TypeOf	ObjectProperty
End	/owl

Tabla 7. Equivalencias entre CARL y OWL-DL

Con esta información, el intérprete es capaz de traducir las sentencias CARL y enviarle al razonador los datos necesarios para que éste pueda ejecutar las consultas adecuadas.

Cuando el razonador finaliza su proceso, envía los datos obtenidos al motor. Éste estudia si existe algún conflicto entre los formatos de los campos que se quieren intercambiar y qué tipo de conflicto es. Si es éste el caso, el motor se encarga de ejecutar la función de transformación correspondiente para solucionar la incompatibilidad y asegurar el intercambio correcto de información.

Como previamente se ha detallado, las funciones de conversión existentes en el sistema se encargan de solucionar los problemas surgidos a nivel de incompatibilidades en el formato de los datos que se desean intercambiar entre las aplicaciones. Esta actividad se puede llevar a cabo gracias a la información recogida en la ontología y a las restricciones establecidas desde el inicio en cuanto a los formatos, fijados también mediante la ontología del dominio. Las funciones de conversión se encuentran en el motor y son capaces de realizar conversiones de datos entre todos los tipos “compatibles” aceptados en el sistema. Al encontrarse estos campos previamente fijados, es posible conocer todos los existentes en el sistema y producir una función de conversión para cada combinación que se pueda dar entre ellos. Estas conversiones se podrán realizar siempre y cuando se encuentren relacionados con el mismo campo de la ontología del dominio, es decir, que se correspondan con un mismo concepto del vocabulario GoodRelations. La Figura 42 muestra algunos ejemplos que representan gráficamente qué tipo de conversiones se pueden realizar y cuáles no debido a discordancias con la ontología del dominio o con los propios formatos de los datos:

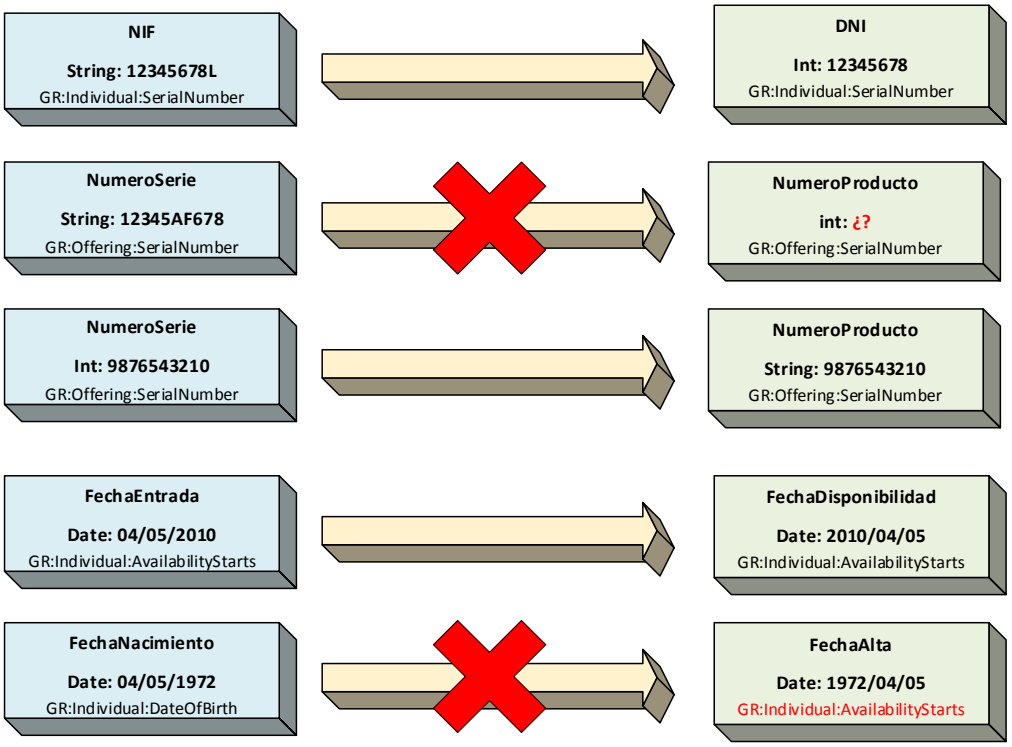


Figura 42. Ejemplos de posibles funciones de conversión

Como se puede observar se pueden dar diferentes casos a la hora de realizar las funciones de conversión que determinarán si se puede ejecutar la transformación o por el contrario existen aspectos que lo impiden. A continuación se explican los ejemplos presentados en la figura con más detalle:

- En el primer ejemplo, es posible realizar la conversión entre NIF y DNI debido a que se encuentran relacionados por el mismo campo de GR y a que la conversión entre ambos no supone una pérdida de información, ya que el DNI no precisa de la letra para su caracterización.

- El segundo ejemplo muestra la imposibilidad de transformar una cadena de caracteres a número ya que, aunque se encuentran relacionados con el mismo campo de GR, se produciría una pérdida de información en el proceso. Esta transformación sí podría llevarse a cabo para pasar de DNI a NIF ya que existe un algoritmo que permite extraer la letra en función de los valores numéricos.

- El tercer ejemplo refleja la situación en la que se pasa de un número a una cadena de caracteres. Como es bien sabido, esta transformación es posible aunque no se perciban cambios en el contenido, ya que un valor numérico tiene la misma representación como cadena de caracteres.

- El cuarto ejemplo realiza transformaciones entre diferentes formatos de fecha aceptados en el sistema. Existen diferentes combinaciones predefinidas que los propietarios podrán seleccionar en cada caso y se permitirá la conversión entre todos los tipos existentes.

- El quinto y último ejemplo muestra la situación en la que, a pesar de que se trata de un tipo de datos compatible, no se encuentra relacionado con el mismo campo en GR. Esto significa que se estaría realizando la transformación entre datos que no se refieren a lo mismo y que, de permitirse, podrían causar problemas en el sistema. Es por ello que nunca se llegará a esta situación ya que las operaciones realizadas mediante el razonador impedirán que se puedan relacionar conceptos no relacionados por el mismo campo en la ontología del dominio, si bien es interesante comentar el comportamiento del sistema.

Explicadas las diferentes posibilidades que pueden darse en cuanto a las conversiones, la Tabla 8 detalla todas las posibles conversiones y las acciones necesarias para llevarse a cabo:

Formato Entrada	Formato Salida	Descripción
Date	**Date**	Se realizan conversiones entre todos los tipos de fecha permitidos en el sistema mediante rotación de la información y sustitución de separadores (si los hubiera).
Cadena	**Número**	Se realiza truncamiento de caracteres siempre y cuando no suponga pérdida de información relevante.
Número	**Cadena**	Se realiza la conversión de tipo sin variación en la información ya que un número también puede representarse como cadena de caracteres.
Tamaño Mayor	**Tamaño Menor**	En campos con restricciones de tamaño o con tamaño predefinido, se reducirá siempre y cuando no conlleve pérdida de información. Este formato es aplicable a números y cadenas de caracteres.
Tamaño Menor	**Tamaño Mayor**	En campos con restricciones de tamaño o con tamaño predefinido, se aumentará siempre el mismo hasta cumplir el requisito. Este formato es aplicable a número y cadenas de caracteres.
Booleano	**Cadena**	Se transformará el valor booleano a su traducción directa como cadena de caracteres.
Cadena	**Booleano**	Se realizará esta transformación siempre que el valor de la cadena coincida con el alguno de los valores booleanos permitidos.
Predefinidos		Los valores predefinidos se tratarán de forma independiente y serán contemplados como tal en el sistema, desarrollando funciones de conversión adecuadas y permitiendo o no su transformación en función del propio formato. Por ejemplo, se podría definir DiaSemana como un tipo que pudiera tomar los valores de los días de la semana. Se podrían realizar funciones de conversión con cadenas de caracteres, pero no con fechas o números.

Tabla 8. Conversiones de datos contempladas por CARL

Las conversiones reflejadas en la tabla anterior muestran valores generales de los datos, de forma que cadenas de caracteres incluyen todos los tipos de datos asociados (String, char, etc.) y los numéricos también a sus respectivos (int, long, short, double, etc.). En estos últimos se les aplicarán las restricciones de tamaño especificadas en la Tabla 8, de forma que sólo se realizarán las transformaciones cuando no se produzca pérdida de información. En el caso de números decimales a enteros se aplicará truncamiento, eliminando las cifras decimales correspondientes.

Una vez que el motor ha realizado las conversiones correspondientes tras recibir la información del razonador, éste remite los datos requeridos en el formato deseado para que sean enviados de vuelta a la aplicación que los solicitó. Esta operación se realiza a través del Interoperability Interface, con la que finaliza el proceso consiguiendo la interoperabilidad automática y dando por terminadas las funciones del motor hasta la llegada de una nueva petición.

Como se ha podido comprobar a lo largo de la sección, la funcionalidad de este componente es clave ya que permite manejar las distintas situaciones que se suceden en los procesos de comunicación, gestionando las diferentes fases y módulos por los que debe pasar el mensaje CARL, facilitando su entendimiento y transformación, alcanzando la interoperabilidad como resultado final.

4.4 Ontología del dominio

Para optimizar el funcionamiento y los contenidos presentados al usuario, se decidió investigar sobre la posibilidad de construir una ontología del dominio que proporcionase los términos y campos necesarios para cubrir el ámbito de acción del sistema.

Profundizado en este ámbito, se comprobó la existencia de varios vocabularios ya construidos que podían realizar la función mencionada con la ventaja de saber que estos ya se encuentran extendidos y en ciertos casos validados por parte de la comunidad.

A continuación se presenta un estudio de las principales opciones encontradas y que pueden adaptarse a los requisitos necesarios.

4.4.1 Alternativas encontradas

Como se ha comentado con anterioridad, los lenguajes y las ontologías se han empleado en multitud de ámbitos y dominios. Actualmente sigue existiendo gran actividad en torno este campo dada su utilidad y el nuevo interés suscitado por la semántica debido a la “reciente” aparición del paradigma de Linked Data.

La existencia de lenguajes y ontologías de dominio es un hecho. En numerosos proyectos e investigaciones se hace necesaria introducir esta figura que cubra el ámbito necesario en cada caso y permita mantener una correcta estructura funcional y conceptual. Para el caso del que se ocupa esta investigación se hace necesaria la presencia de una ontología del dominio que sea capaz de capturar el conocimiento necesario para una plataforma de Cloud Computing completamente heterogénea y abierta en cuanto a contenidos se refiere (no se ciñe a un dominio en particular). Por ello se hace necesario conseguir un vocabulario general, que de cabida a conceptos de áreas dispares y pueda integrarse para trabajar conjuntamente con CARL.

A lo largo de los siguientes apartados se presentan las opciones más factibles encontradas, incluyendo información detallada sobre ellas. Seguidamente se presenta una discusión posterior al análisis de alternativas que llevará a la elección de una de ellas para convertirse en ontología del dominio del sistema.

4.4.1.1 DBpedia

DBpedia es un conocido proyecto desarrollado por la Universidad de Leipzig, la Universidad Libre de Berlín y la compañía OpenLink Software que pretende extraer información de Wikipedia para crear una versión semántica de la misma (Auer et al., 2007). Por tanto, su objetivo es el de estructurar la información de Wikipedia y hacerla accesible a través de la Web. De esta manera se hace posible realizar consultas complejas (a través de SPARQL) y unir diferentes conjuntos de datos de la Web con información de Wikipedia.

Las bases de conocimiento han ido incrementando su presencia e importancia durante los últimos años a la hora de gestionar la inteligencia de la Web y en los procesos de integración de información. Actualmente, la mayoría de bases de conocimiento se centran en dominios específicos, son desarrollados por pequeños grupos de investigadores y conllevan un alto coste a la hora de mantener actualizados los cambios que se producen en el dominio en cuestión. Al mismo tiempo, Wikipedia se ha convertido en uno de los centros de conocimiento más relevantes, mantenido por miles de colaboradores.

DBpedia hace uso de esta enorme fuente de conocimiento para extraer información estructurada de Wikipedia y la hace accesible a través de la Web bajo los términos de la Creative Commons Attribution-ShareAlike 3.0 License y el GNU Free Documentation License. La versión en inglés de la base de conocimiento de DBpedia describe actualmente 3.77 millones de términos, de los cuales 2.35 millones se encuentran clasificados en una ontología consistente que incluye 764.000 personas, 573.000 lugares, 330.000 trabajos creativos (música, películas, videojuegos, etc.), 192.000 organizaciones, etc. Además, se proporcionan varias versiones en 111 idiomas que juntas describen 20.8 millones de términos.

Esta base de conocimiento presenta una serie de ventajas sobre otras bases, como por ejemplo:

- Cubre gran cantidad de dominios
- Representa una aceptación real por parte de la comunidad
- Evoluciona automáticamente con los cambios que se producen en Wikipedia
- Es multilingüe

Por otro lado, el conjunto de datos de DBpedia consiste en 1.89 billones de datos (en tripletas RDF) repartidos en diferentes idiomas y dominios. Cada entidad en Dbpedia se describe a través de varias propiedades: etiqueta, abstract corto y largo en inglés, un link a la correspondiente entrada de la Wikipedia y un link a una imagen que describe el concepto (si se encuentra disponible).

Además proporciona tres tipos diferentes de esquemas de clasificación:

- Las categorías de Wikipedia se representan empleando el vocabulario SKOS (Miles, Matthews, Wilson, & Brickley, 2005) y términos DCMI (Powell, Nilsson, Naeve, & Johnston, 2005).
- La clasificación YAGO (Suchanek, Kasneci, & Weikum, 2007) deriva del sistema de categorías de Wikipedia usando Word Net (Pedersen, Patwardhan, & Michelizzi, 2004).
- Finalmente, se generan Word Net Synset Links (Fernandez, d' Aquin, & Motta, 2011) manualmente relacionando las plantillas de información de Wikipedia con Word Net synsets y añadiendo los correspondientes links a cada cosa que utiliza una plantilla determinada. Teóricamente, este tipo de clasificación debe ser más precisa que la que se proporciona con el sistema de categorización de la Wikipedia.

Como último elemento clave de DBpedia se encuentra la DBpedia Ontology (Bizer, Lehmann, et al., 2009). Ésta es una ontología superficial, que cubre varios dominios y generada manualmente en función de la información más utilizada en Wikipedia. La ontología posee actualmente 359 clases que forman una jerarquía de subsunción y que está descrita por 1.775 propiedades diferentes y contiene más de 2 millones de instancias. Esta ontología se encuentra en un continuo proceso de actualización y en la última versión ha dejado de tener una estructura de árbol para pasar a ser un grafo acíclico directo. La Figura 43 muestra la estructura aquí mencionada:

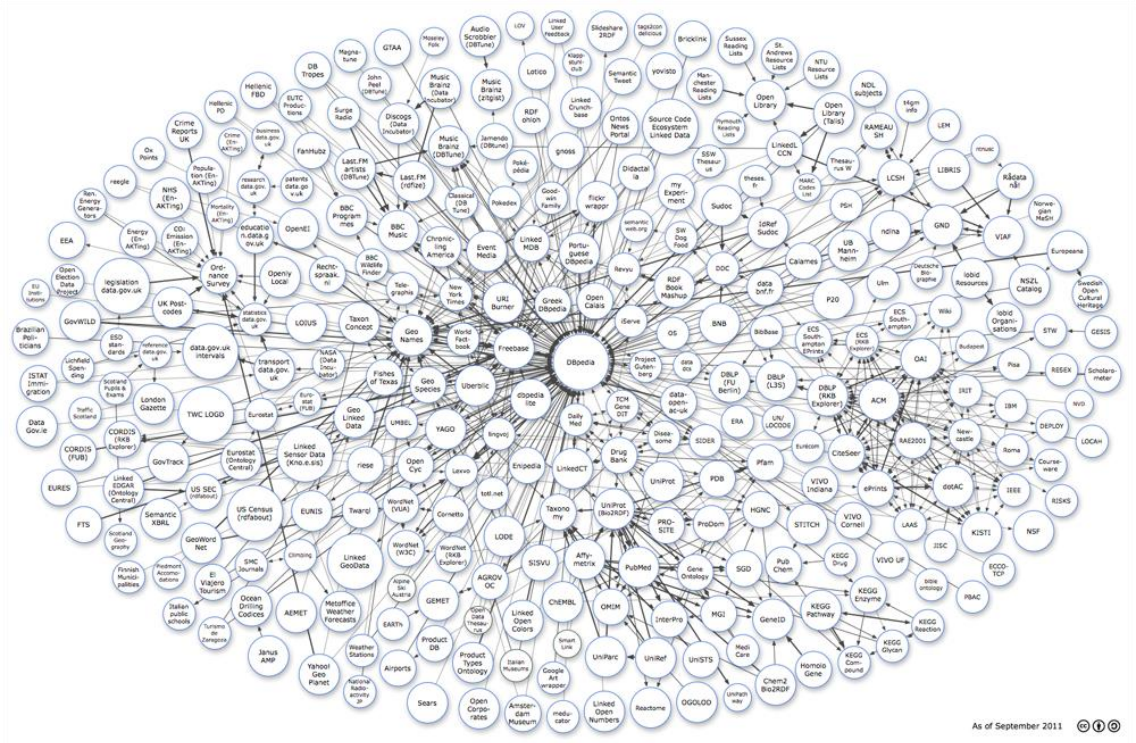


Figura 43. Estructura de DBpedia

Conjuntamente con el esfuerzo de la comunidad del W3C Linking Open Data (LOD), un gran número de proveedores han comenzado a publicar datos de acuerdo a los principios de Linked Data presentados por Tim Berners-Lee. La Web resultante consiste en varios billones de tripletas RDF y la posibilidad de cubrir numerosos dominios (información geográfica, personas, compañías, música, libros, comunidades online, etc.). Además de todo esto, existen trabajos en curso basados en Linked Data como pueden ser por ejemplo navegadores, crawlers o buscadores.

La base de conocimiento de DBpedia se presenta como Linked Data en la Web ya que como DBpedia define URIs de Linked Data para millones de conceptos, varios proveedores de datos han comenzado a colocar links RDF desde sus datos a DBpedia, convirtiéndola en uno de los elementos centrales para la emergente “Web of Data”.

4.4.1.2 Good Relations

GoodRelations se presenta como un vocabulario “estándar” para la realización de transacciones en el entorno del comercio electrónico (Hepp, 2008). Su potencial es enorme, ya que puede ser embebido en páginas web estáticas o dinámicas y puede ser procesado por otros ordenadores.

La generalidad del diseño del vocabulario (o esquema y ontología) permite que sea aplicado a entornos que no se ciñan de manera estricta al dominio del e-commerce y que se plantee como una alternativa estándar y soportada de manera general al sistema del que se ocupa este trabajo.

GoodRelations es un lenguaje que puede utilizarse para describir de forma muy precisa distintos tipos de negocio. El aspecto clave es que se pueden crear pequeños paquetes de datos que describan la totalidad del entorno en el que se aplique, sea cual sea. Las principales características de las que se compone son las siguientes:

- Las anotaciones realizadas mediante GoodRelations son respetadas tanto por los motores de búsqueda tradicionales como por los novedosos basados en Linked Data/Aplicaciones de Web Semántica.
- Multisintaxis: Los datos pueden ser publicados como RDF en HTML, Microdata, RDF/XML, etc.
- Mínimo impacto del tamaño de página y tiempo de carga.
- Incluye información de la compañía y toda serie de conceptos en torno a sus actividades.
- Niveles de detalle variables en función de los requisitos del negocio.
- Adecuado para escenarios B2C y B2B.
- Puede combinarse con el protocolo Open Graph de Facebook.

Estas características hacen del enfoque una opción sencilla a la par que eficaz para representar el conocimiento de todo tipo de negocios. Además la ontología se encuentra en continuo desarrollo y se actualiza con nuevas clases y propiedades que puedan resultar de interés para el cliente.

Con tofo ello, los objetivos definidos por GoodRelations se centran en definir una estructura de datos para el comercio electrónico que sea:

- **Industrialmente neutral**, es decir, adecuada para cualquier tipo de bienes o servicios
- **Válido para cualquier etapa de la cadena de valor**
- **Sintácticamente neutral**. Trabaja en microdatos, RDFa, RDF/XML, Turtle, JSON, OData, GData y otras sintáxis populares.
- **Trabaja con mínimo razonamiento.**
- Usable en entornos de Web Semántica del W3C, pero no depende de ello.
- **OWL-DL**: puede mezclarse con otros esquemas OWL-DL y datos sin llegar a un resultado de OWL Full.
- **Extensible.**
- Granularidad dinámica de los datos.
- Refinamiento incremental.

Estos objetivos se logran mediante la utilización de cuatro entidades que representan escenarios del comercio electrónico:

- **Un agente**: Persona u organización.
- **Un objeto o servicio**: Una cámara, una casa, un corte de pelo.
- **Una promesa, producto u oferta**: Transfiere algunos derechos sobre el objeto o servicio proporcionado por una compensación determinada (por ejemplo, dinero) por parte del agente al objeto concreto.
- **Una localización**: Desde la que la oferta está disponible (almacén, parada de autobús…)

El principio de Agente-Oferta-Objeto se encuentra en la mayoría de las industrias y es la base del potencial genérico de GoodRelations, ya que permite representar conceptualmente con el mismo vocabulario la oferta de un teléfono móvil, un tratamiento de belleza o una reparación del coche.

Como se puede comprobar, con un conjunto sencillo de entidades se puede representar un gran conjunto de información, sin embargo, como se muestra en la Figura 44, su diseño sencillo no excluye la existencia de un amplio espectro de representación, simplemente saca el máximo partido de los conceptos representados.

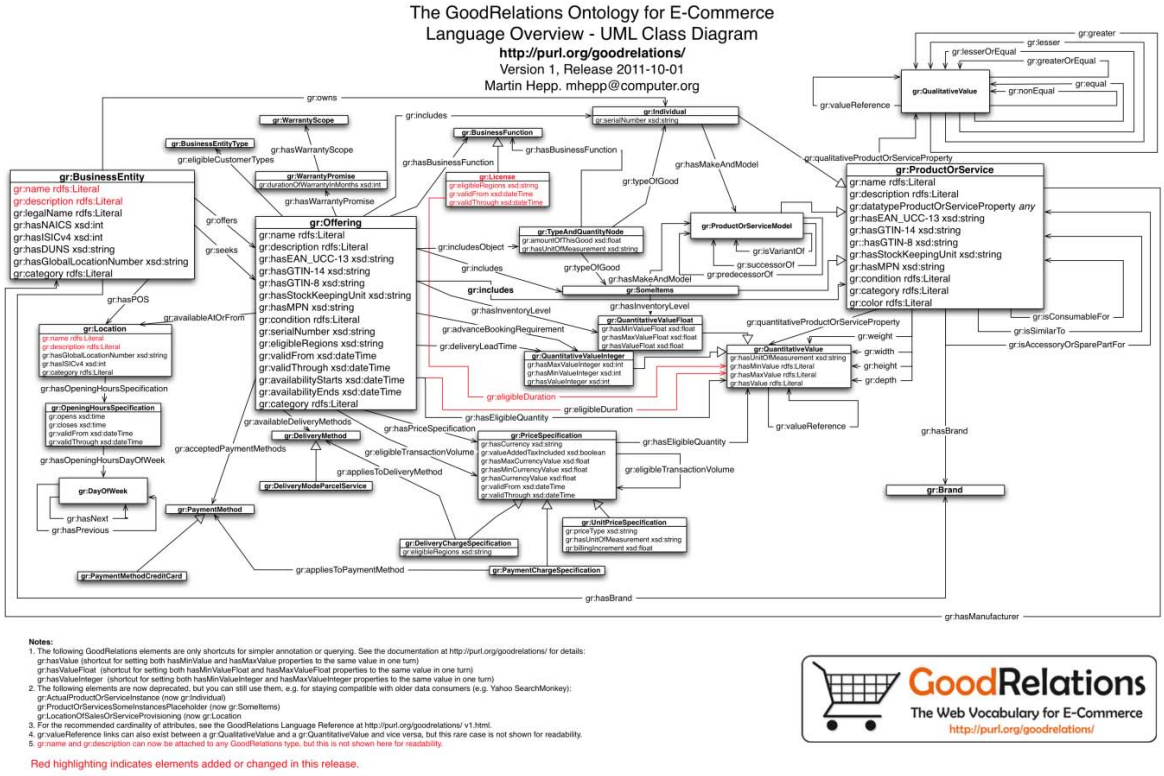


Figura 44. Ontología GoodRelations

Como es posible comprobar, el potencial de este enfoque es enorme y esto ha hecho que grandes compañías como Google o Yahoo! recomienden este vocabulario para el manejo y envío de información estructurada.

4.4.2 Discusión

Las dos alternativas planteadas representan las opciones más adecuadas para cubrir las necesidades del sistema. Existen otros lenguajes, ontologías y vocabularios válidos, pero la mayoría de ellos se encuentran centrados en entornos concretos lo que limitaría el alcance de la plataforma y restaría valor e importancia a la aportación de CARL en los procesos de captura del conocimiento y en los de comunicación de aplicaciones.

DBpedia ha alcanzado una notable relevancia en la comunidad investigadora debido a que su éxito se encuentra unido al de Wikipedia y muy relacionado con el auge del paradigma de Linked Data. Cuenta con el apoyo de organismos importantes en la comunidad y sin duda representa una manera adecuada de representar contenido en la Web. Sin embargo, su amplitud la hace demasiado extensa para los objetivos que debe cumplir. No se debe perder de vista el objetivo del sistema, que es el de ser capaz de proporcionar información sobre el dominio que, aunque en este caso no se encuentre acotado, no precisa de información tan amplia como la que este vocabulario proporciona, lo que puede llegar a convertir la solución en un problema.

Por tanto, la solución elegida para actuar como ontología del dominio será GoodRelations ya que presenta las características más adaptadas en relación a los objetivos de la investigación. Su aparición es más reciente a la de DBpedia y aún así, compañías fuertes en el sector apuestan por la versatilidad y sencillez del trabajo coordinado por Martin Hepp. Sus características resultan interesantes para aplicarse a multitud de dominios y esto, al estar dirigido a una plataforma de aplicaciones heterogéneas de Cloud Computing, supone una ventaja definitiva. Su generalidad y capacidad para combinar entidades resultará muy útil cuando se integre en el sistema global presentado. De esta manera, GoodRelations proporcionará la información correspondiente al dominio y ajustará los datos introducidos por los usuarios en la plataforma de computación en nube a las clases contenidas en su ontología para conseguir así un común entendimiento entre ellas y dar el siguiente paso hacia la interoperabilidad.

4.5 Diseño de la base de datos

Uno de los aspectos clave en el diseño de sistemas y plataformas de Cloud Computing reside en la forma en la que se configura la base de datos. En entornos multiusuario es de vital importancia poseer un modelo de base de datos que sea capaz de recoger y dar servicio a los diferentes tenants asegurando la seguridad de acceso sin incurrir en unos elevados costes de gestión y mantenimiento. Esta tarea que en un principio puede no parecer complicada puede afrontarse desde distintas perspectivas en función de las necesidades del sistema concreto y de las prioridades que se le quiera otorgar.

A lo largo de esta sección se proporciona información acerca de las opciones más interesantes en cuanto a diseño de bases de datos en arquitecturas multitenant se refiere y que permitirá asegurar una correcta persistencia de la información recogida en el sistema. Una vez estudiadas las ventajas e inconvenientes de cada una de estas posibilidades se procede a la posterior elección y justificación de la opción más adecuada para el caso objeto de esta investigación.

4.5.1 Alternativas de diseño

La implementación de arquitecturas multitenant requiere un rediseño de la estructura de la base de datos para acomodar los datos de varios tenant en una misma estructura. Esto puede conseguirse de múltiples formas, entre las que destacaremos, ordenadas de mayor a menor aislamiento:

- Una base de datos por tenant
- La misma base de datos pero un esquema para cada tenant
- Esquema compartido por todos los tenant

Figura 45. Nivel de aislamiento en la configuración de bases de datos (Microsoft)

4.5.1.1 Una base de datos por tenant

Esta es, probablemente, la forma más sencilla de almacenar los datos de diferentes tenant. Consiste en mantener una base de datos (aunque sea dentro del mismo servidor) para cada uno mientras que los recursos computacionales (servidores de aplicación) son compartidos (ver Figura 46).

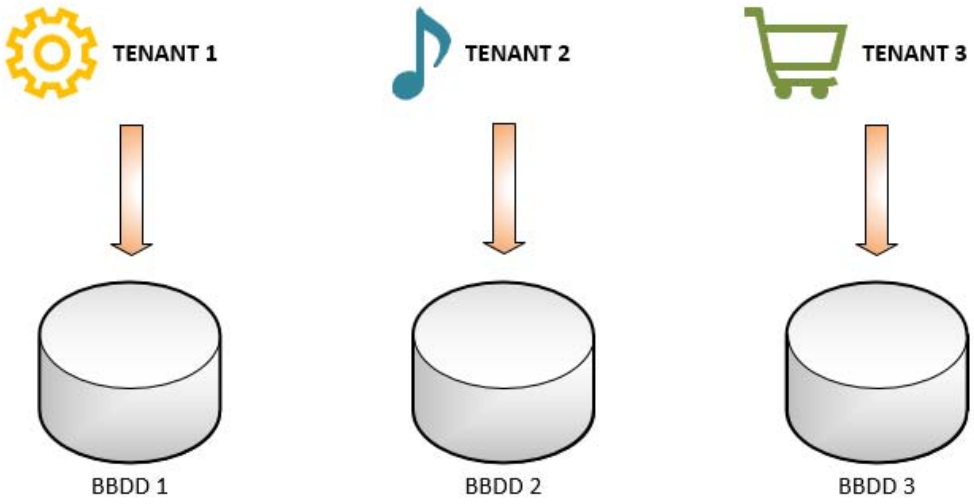


Figura 46. Diferentes bases de datos para cada tenant

La implementación de esta arquitectura suele requerir cambios mínimos en la aplicación ya que el comportamiento es el mismo que para un solo usuario excepto por tener datos de conexión a la base de datos distintos en función del tenant. Además, facilita las extensiones particulares a un sólo tenant (ya que no requiere cambio ninguno a los datos del resto) y restaurar las copias de seguridad de un sólo tenant no tiene complicaciones especiales.

Sin embargo, los costes de hardware y mantenimiento suelen ser mayores que los de otras soluciones con menos aislamiento. Esto es debido a que precisa de un mayor espacio de almacenamiento, mayor cantidad de servidores, etc. Además, el número de tenants en la aplicación puede verse limitado por el número máximo de bases de datos gestionables por el motor.

Por todo esto, esta solución suele usarse cuando el aislamiento de los datos o la personalización extrema por usuarios son características importantes, por lo que merece la pena el gasto adicional en el que se incurre al usar esta aproximación.

4.5.1.2 Un esquema por tenant

Al igual que en el caso anterior, cada tenant tiene un conjunto de tablas diferentes, pero en lugar de encontrarse en bases de datos separadas, lo hacen en distintos esquemas de la misma base de datos.

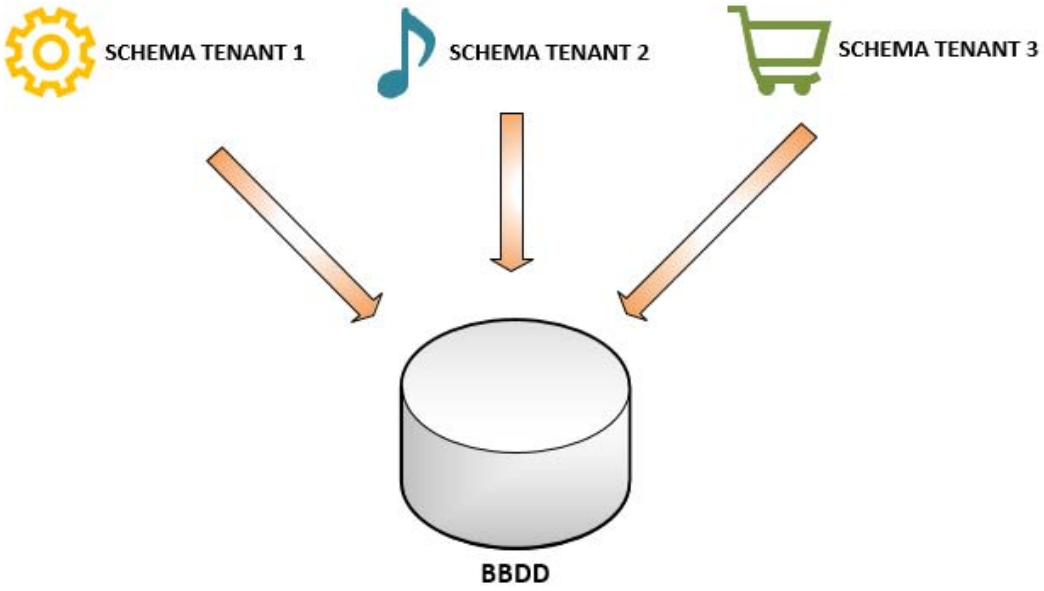


Figura 47. Misma base de datos, diferentes esquemas para cada tenant

Como en el caso anterior, los cambios en la aplicación son mínimos. Una posible opción, muy similar al sistema de bases de datos separadas, es crear un usuario de base de datos para cada tenant que tenga como esquema predeterminado el correspondiente al tenant, de modo que sin modificarse las consultas se dirijan directamente a las tablas necesarias. También se pueden modificar las consultas para incluir el esquema de forma explícita de manera de que toda la aplicación pueda compartir una única conexión a la base de datos.

Con esta solución, los costes son menores que con la anterior, ya que al compartirse la misma base de datos entre múltiples usuarios los recursos de hardware necesarios disminuyen. Esta opción soporta más usuarios que la anterior, ya que el número máximo de esquemas en una base de datos suele ser ilimitado, a diferencia del número de bases de datos por servidor.

En cambio, la restauración de la base de datos es más complicada, ya que si el fichero de copia de seguridad no incluye información que permita restaurar un sólo esquema es necesario restaurarlos todos, afectando a todos los tenant aunque no hayan perdido datos. Tampoco se consigue el mismo nivel de aislamiento que con bases de datos diferentes.

Esta solución es apropiada para aplicaciones con no demasiadas tablas (menos de 100) en las que el aislamiento no sea fundamental y los costes de funcionamiento sean un factor importante.

4.5.1.3 Esquema compartido

Esta tercera solución utiliza las mismas tablas para almacenar los datos de todos los tenant, separándolos mediante un campo en la tabla que indique a qué tenant pertenece cada uno de los datos almacenados.

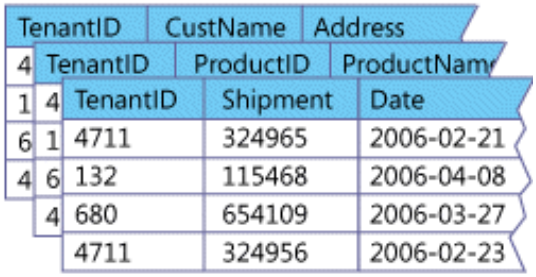


Figura 48. Compartición de esquema de base de datos (Microsoft)

A nivel de aplicación, este sistema implica el mayor número de cambios, ya que hay que modificar las consultas para incluir un filtro por identificador de tenant para mostrar sólo la información del perteneciente al cliente que esté lanzando la petición.

Esta arquitectura tiene menores costes de mantenimiento y hardware que las anteriores, ya que la compartición de la infraestructura de esta forma permite servir más tenant por servidor de base de datos. Sin embargo, debido a la mayor complicación a nivel de aplicación, los costes iniciales de desarrollo son mayores.

La restauración de copias de seguridad genéricas es mucho más complicada en esta implementación si es necesaria una restauración de un sólo tenant, debido a que los datos a restaurar son filas específicas de una tabla. Una posible solución es utilizar un sistema de copias de seguridad propio, que exporte distintos ficheros según el tenant. Esto simplifica la restauración, aumentando aun más los costes de desarrollo.

Esta solución es la más adecuada cuando los costes son un factor determinante de la aplicación o es necesario servir a un gran número de tenants con pocos servidores y los clientes están dispuestos a renunciar al aislamiento de datos a cambio de costes menores.

4.5.1.4 Discusión

Cada una de las aproximaciones expuesta presenta una serie de ventajas e inconvenientes que hacen de cada una de ellas la opción apropiada en unos casos u otros.

En primer lugar se encuentran los factores económicos. Como se comentaba anteriormente, el enfoque compartido requiere de una mayor inversión inicial debido a la complejidad de desarrollo de arquitecturas compartidas, sin embargo, al soportar un mayor número de tenants por servidor, a largo plazo el coste se reduce de manera considerable.

Otro aspecto a tener en cuenta es la seguridad. Existe una idea equivocada en torno a esto acerca de que sólo el aislamiento físico puede proporcionar un nivel adecuado de seguridad, de hecho, sistemas que almacenan los datos empleando enfoques compartidos pueden proporcionar altos grados de seguridad mediante la utilización de algunos patrones algo más sofisticados.

Como último y clave elemento se encuentran los propios tenants. Su número, naturaleza y necesidades puede orientar la elección de los enfoques de manera determinante. La Figura 49 muestra la influencia de cada uno de los parámetros en la toma de decisiones del modelo a desarrollar:

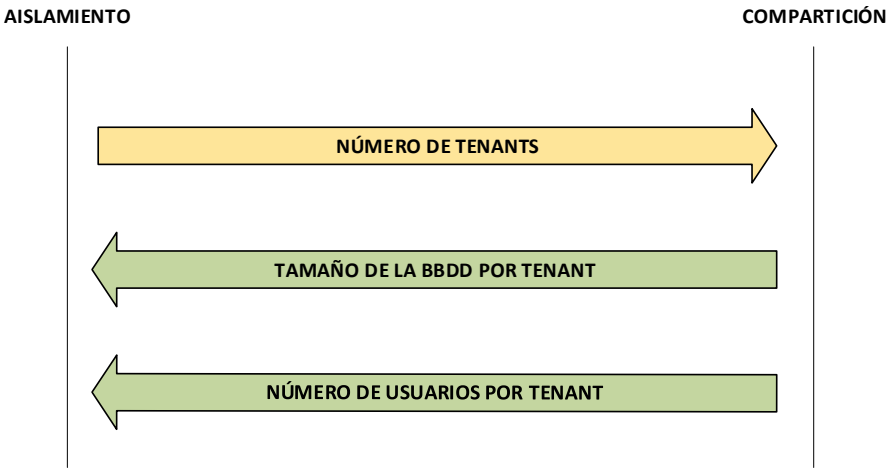


Figura 49. Factores de influencia de Aislamiento vs. Compartición

Una vez conocidos los factores y su influencia se ha decidido que el modelo de base de datos a desarrollar sea el de esquema compartido. Con ello, a pesar de que será necesaria una inversión inicial algo mayor, se obtienen unos beneficios que no proporcionan las otras opciones. Debido a situarse en una estructura de Cloud Computing, se reduce considerablemente el coste de hardware, ya que el paradigma nos permitirá utilizar sólo el espacio de datos que se necesite y aumentarlo dinámicamente según las necesidades concreta de cada momento. Con ello, al tener una única base de datos para todos se elimina el problema de alto coste.

Un problema asociado a este sistema es la personalización. Las plataformas de Cloud Computing basan gran parte de su potencial en la posibilidad de personalización de las aplicaciones por parte de los propietarios, que en cada caso pueden requerir de más o menos información para su funcionamiento en función del negocio y de su visión de él. Una solución bastante sencilla al problema es la de asociar un número fijo de campos de personalización para cada tenant de forma que puedan usarlos de la manera que crean conveniente en cada caso. Bien es cierto que es posible que esto provoque que haya espacio sin utilizar y eso derive en un aumento del espacio físico y por lo tanto de los costes, sin embargo, es una solución aceptable y sencilla de implementar.

Por otro lado, se ha priorizado el número de tenants por encima de los otros parámetros debido a que el sistema a desarrollar está más enfocado a dar servicio a un elevado número de tenants, mientras que los usuarios por cada uno de ellos o los tamaños de sus bases de datos son aspectos gestionables y menos problemáticos gracias a las características de los entornos de computación en nube.

A continuación se proporciona información más detallada de cuál ha sido el diseño final de la base de datos y que modificaciones sobre el modelo "tradicional" se han realizado para dar cabida a las funcionalidades que debe recoger el sistema.

4.5.2 **Diseño final**

Tras el estudio de alternativas realizado, se ha decidido emplear un modelo de esquemas compartidos que permita representar la información de cada tenant aprovechando al máximo el espacio de almacenamiento. En torno a esto surge el problema, previamente explicado, de los campos previamente definidos para la personalización de las aplicaciones. Sin embargo, existe una implementación que permite solucionar este problema a través de una extensión del modelo de datos.

Esta solución implica la inclusión de una tabla en la que se almacenan los datos personalizados y la utilización de metadatos para definir las etiquetas y los tipos de datos para los campos personalizados de cada tenant. Un ejemplo del diseño de esto se puede encontrar en la Figura 50:

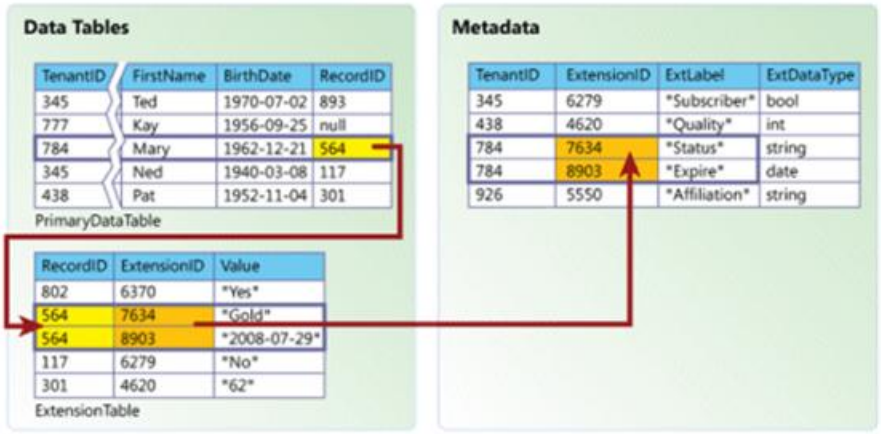


Figura 50. Ejemplo de modelo extensible de base de datos (Microsoft)

Con la implementación de este modelo se evita el inconveniente comentado y se consigue un mejor aprovechamiento del espacio de almacenamiento. De esta forma se proporciona una base de datos con la información de todos los usuarios y tenant así como de las aplicaciones. Consiste principalmente de un modelo de datos extensible que permite añadir un número indefinido de campos personalizados a cada usuario. También hay tablas para guardar la información sobre las aplicaciones y los usuarios con permisos para utilizarlas. El esquema de la base de datos resultante se muestra en la siguiente figura:

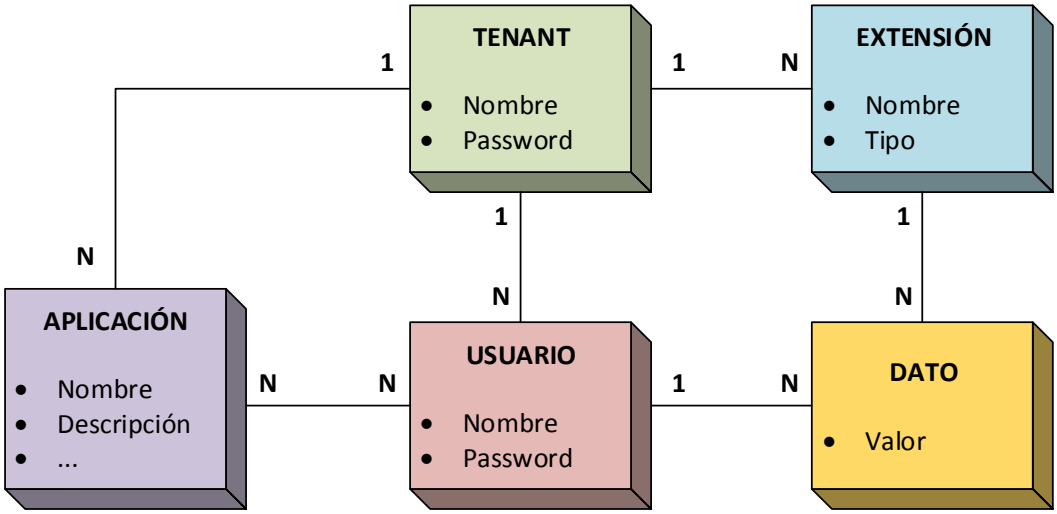


Figura 51. Esquema de la base de datos extensible

En la Figura 51 se muestran las siguientes tablas:

1. *Tenant*: Almacena la información de los tenant: nombre de usuario, contraseña y cualquier dato administrativo necesario (Nombre, NIF/CIF, etc.).
2. *Usuario*: Información básica de los usuarios: nombre de usuario, contraseña y tenant al que pertenecen.
3. *Aplicación*: Los datos necesarios sobre una aplicación, entre los que destacan su nombre y descripción, y la ruta de acceso a la aplicación. Tiene una relación de pertenencia a un tenant (que puede tener varias aplicaciones) y de disponibilidad con usuarios (un usuario puede tener disponibles varias aplicaciones, una aplicación puede estar disponible a varios usuarios).
4. *Extensión*: Descripción de los campos personalizados de un tenant para sus usuarios, en concreto: nombre del campo y tipo de datos (número entero, número real, cadena de caracteres, fecha, etc.).
5. *Dato*: Contenido (valores) de la tabla virtual formada por los usuarios y extensiones de un tenant. Se almacena el valor de cada par usuario-extensión que pertenece al mismo tenant, siempre y cuando no esté vacío.

Este enfoque permite a cada tenant crear tantos campos personalizados como necesite para cumplir sus necesidades de negocio. Cuando una aplicación recupera los datos de un cliente, realiza una búsqueda en la tabla de extensión, selecciona todas las filas correspondientes al identificador y devuelve un valor por cada campo personalizado utilizado. Para asociar estos valores con los campos adecuados, la aplicación realiza otra búsqueda en los metadatos usando los identificadores asociados en la tabla de extensión.

Empleando este modelo extensible de datos se consiguen los objetivos perseguidos: una utilización eficiente de los recursos, una configuración adecuada a las metas y la persistencia de la información de usuarios, tenants y aplicaciones a través de una base de datos específica para arquitecturas multiusuario.

4.6 Infraestructura de Cloud Computing

Un aspecto importante para el desarrollo de las actividades objeto de esta investigación, es la selección de una plataforma de Cloud Computing que ofrezca los servicios necesarios para soportar las diversas acciones y funcionalidades que se llevarán a cabo en el sistema.

Actualmente, y debido al gran éxito cosechado por este paradigma en tan poco tiempo, han surgido un gran número de compañías que ofertan servicios de computación a través de Internet y otras muchas que se encargan de proporcionar infraestructuras para que se puedan desarrollar plataformas sobre ellas. Para los objetivos que plantea esta tesis doctoral se hace necesario encontrar una compañía que proporcione IaaS, sobre la cual se desarrolle el sistema planteado donde actuará el lenguaje y el resto de componentes que complementan su acción (que formarán un PaaS semántico) y sobre el cual los tenants subirán los servicios que serán consumidos por los usuarios finales.

A continuación se presentarán una serie de empresas que constituyen las alternativas sobre las que se elegirá la plataforma sobre la que desplegar el sistema.

4.6.1 Alternativas estudiadas

Debido a las numerosas empresas que hoy en día proporcionan estas infraestructuras, se ha decidido seleccionar tres de ellas para su estudio y análisis posterior. Se trata de tres de las compañías punteras y pioneras del sector, que han llevado sobre sus hombros gran parte de la evolución de las tecnologías asociadas al Cloud Computing. Los IaaS estudiados serán Amazon Elastic Compute Cloud, Windows Azure y Google App Engine. Con posterioridad a su estudio se realizará una discusión para elegir la infraestructura que mejor se adapta a los requisitos del sistema diseñado.

4.6.1.1 Amazon Elastic Compute Cloud (Amazon EC2)

Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona capacidad informática en la nube con tamaño modificable. Está diseñado para facilitar a los desarrolladores recursos informáticos escalables y basados en web.

Amazon EC2 reduce el tiempo necesario para obtener, permitiendo arrancar nuevas instancias de servidor en minutos, lo que permite escalar rápidamente la capacidad según cambien sus necesidades. Presenta además un auténtico entorno informático virtual, que permite utilizar interfaces de servicio web para crear instancias con distintos sistemas operativos, cargarlas

con un entorno de aplicaciones personalizado, gestionar permisos de acceso a la red y ejecutar la imagen utilizando los sistemas que se elija.

Como una de las características clave se puede decir que EC2 ha sido pionero en el cambio el modelo económico tradicional en informática, permitiendo pagar sólo por la capacidad que se utiliza realmente. Este hecho le ha situado como uno de los grandes pilares de este nuevo paradigma informático.

4.6.1.1.1 Características Principales

Como se comentaba previamente, Amazon fue una de las primeras grandes empresas en lanzarse al universo del Cloud Computing, ofreciendo una serie de servicios e infraestructuras que nadie había ofrecido hasta el momento. Dentro de este conjunto de posibilidades, la plataforma de EC2 posee una serie de características principales que lo identifican dentro del mercado de la computación en nube:

- **Elástico**. Amazon EC2 permite aumentar o reducir la capacidad en cuestión de minutos. Puede enviar una, cientos o incluso miles de instancias del servidor simultáneamente. Desde luego, como toda operación que se controla con un API de servicio web, la aplicación se escalará (aumentará o disminuirá su capacidad) dependiendo de sus necesidades.

- **Control total**. Control total sobre las instancias. Se posibilita el acceso de usuario raíz (root) y la interacción con ellas como con cualquier otra máquina. Se pueden detener las instancias y mantener los datos en la partición de arranque, para reiniciar a continuación la misma instancia a través de las API del servicio web. Las instancias se pueden reiniciar de forma remota mediante las API del servicio web. Asimismo, se tiene acceso a la emisión de consola de las instancias.

- **Flexible**. Posibilidad de elegir entre varios tipos de instancia, sistemas operativos y paquetes de software. Amazon EC2 permite seleccionar una configuración de memoria, CPU, almacenamiento de instancias y el tamaño de la partición de arranque óptimo para su sistema operativo y su aplicación. Por ejemplo, entre sus opciones de sistemas operativos se incluyen varias distribuciones de Linux y Microsoft Windows Server.

- **Diseño pensado para su uso con otros Amazon Web Services.** Amazon EC2 trabaja con Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), Amazon SimpleDB y Amazon Simple Queue Service (Amazon SQS) para proporcionar una solución informática completa, incluyendo procesamiento de consultas y almacenamiento en una gran gama de aplicaciones.

- **Fiable.** Amazon EC2 ofrece un entorno muy fiable en el que las instancias de sustitución se pueden enviar con rapidez y anticipación. El servicio se ejecuta en los centros de datos y la infraestructura de red acreditados de Amazon. El compromiso del

contrato a nivel de servicio de Amazon EC2 es de una disponibilidad del 99,95% en cada Región de Amazon EC2.

- **Seguro.** Amazon EC2 funciona junto con Amazon VPC (Virtual Private Cloud) para proporcionar una funcionalidad de la red sólida y segura.

- **Económico.** Amazon EC2 permite disfrutar de las ventajas financieras de Amazon. Ofrece una tarifa muy baja por la capacidad informática que realmente utiliza. Según el tipo de tarifa, hay diferentes tipos de instancias para comprar:

    - *Instancias en demanda*: Con On-Demand Instances se puede pagar por la capacidad informática por hora, sin compromisos a largo plazo. Esto libera al cliente de los costes y las complejidades de la planificación, la compra y el mantenimiento del hardware y transformará lo que normalmente son grandes costes fijos en costes variables mucho más pequeños. Gracias a On-Demand Instances también se elimina la necesidad de comprar una "red de seguridad" de capacidad para gestionar picos de tráfico periódicos.

    - *Instancias reservadas*: Las instancias reservadas ofrecen la opción de realizar un pago puntual reducido por cada instancia que se desea reservar, y recibir a cambio un descuento importante en el cargo de uso por horas de dicha instancia. Existen tres tipos de instancias reservadas (instancias reservadas de utilización ligera, mediana e intensa) que permiten equilibrar el importe del pago anticipado a realizar con su precio por hora efectivo. Amazon pone a disposición de sus clientes el Marketplace de instancias reservadas, que ofrece la oportunidad de vender instancias reservadas si cambian las necesidades.

    - *Instancias puntuales:* Con las instancias puntuales, los clientes pueden ofertar la capacidad sin utilizar de Amazon EC2 y ejecutar dichas instancias mientras su oferta supere el precio puntual. El precio puntual cambia periódicamente según la oferta y la demanda, y los clientes cuyas ofertas alcancen o superen dicho precio tendrán acceso a las instancias puntuales disponibles. Si se es flexible respecto a cuándo ejecutar las aplicaciones, las instancias puntuales pueden reducir significativamente los costes de Amazon EC2.

    Estos tipos (demanda, reservada y puntual), se combinan con los diferentes tipos de instancias según su funcionalidad para obtener el precio final.

4.6.1.1.2 Servicios

Dentro de su configuración, Amazon EC2 ofrece una serie de servicios o aplicaciones que permiten completar la experiencia de la computación en nube con propiedades en algunos casos necesarias y en otros con un carácter de complementariedad. A continuación se presenta un resumen de las más relevantes.

Amazon CloudWatch (autoescalabilidad)

Proporciona la supervisión de los recursos de la nube de Amazon Web Services (AWS) y de las aplicaciones que los clientes ejecutan en AWS. Los desarrolladores y administradores de sistema la utilizan para recopilar métricas y realizar su seguimiento, obtener conocimientos y reaccionar inmediatamente para que sus aplicaciones y empresas sigan funcionando sin problemas. Amazon CloudWatch supervisa recursos de AWS como las instancias de bases de datos de Amazon EC2 y Amazon RDS, y también puede supervisar métricas personalizadas generadas por las aplicaciones y los servicios de un cliente.

Este servicio web permite visualizar la utilización de recursos, el funcionamiento operativo y los patrones de demanda en general (incluido el uso de CPU, las operaciones de lectura y escritura en disco y el tráfico de red). Asimismo, obtiene estadísticas, gráficos y define alarmas para datos métricos.

Blueprints

Amazon denomina "AMIs" a sus blueprints o imágenes para acelerar y facilitar el aprovisionamiento de instancias en la nube. Una máquina de imagen Amazon (AMI) es un tipo especial de sistema operativo pre-configurado y software de aplicaciones virtualizadas que se utiliza para crear una máquina virtual en EC2. Además, es la unidad básica de la implementación de los servicios prestados mediante EC2.

La plataforma cuenta con más de 2200 imágenes de máquinas virtuales alternativas, con diferentes sistemas operativos, aplicativos y configuraciones. Una de las configuraciones más populares cuenta con un sistema operativo "Ubuntu" y software de base "LAMP" (Linux, Apache, MySQL y PHP). Además, las instancias de AMIs se pueden filtrar por Proveedor.

Soporte para almacenamiento de datos

En este apartado se provee información acerca de las alternativas ofrecidas por la plataforma EC2 para la persistencia de datos:

- **Amazon Simple Storage Service (Amazon S3).** Amazon S3 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. Concede acceso a todos los desarrolladores a la misma infraestructura económica, altamente escalable, fiable, segura y rápida que utiliza Amazon para tener en funcionamiento su propia red internacional de sitios web. Este servicio tiene como fin maximizar las ventajas del escalado y trasladar estas ventajas a los desarrolladores.
- **Amazon Relational Database Service (Amazon RDS).** Amazon Relational Database Service (Amazon RDS) es un servicio web que facilita las tareas de configuración, utilización y escalado de una base de datos relacional en la nube. Proporciona

capacidad rentable y de tamaño modificable y, al mismo tiempo, gestiona las tediosas tareas de administración de la base de datos, lo que le permite centrarse en sus aplicaciones y en su negocio. Permite acceder a todas las funciones de un motor de base de datos MySQL, Oracle o Microsoft SQL Server conocido. Esto supone que el código, las aplicaciones y las herramientas que el cliente ya utiliza en la actualidad con sus bases de datos existentes, funcionarán a la perfección con Amazon RDS.

- **Amazon SimpleDB.** Amazon SimpleDB es un almacén de datos no relacionales de alta disponibilidad y flexible que descarga el trabajo de administración de bases de datos. Los desarrolladores simplemente almacenan elementos de datos y los consultan mediante solicitudes de servicios Web; Amazon SimpleDB se encarga del resto. Sin las limitaciones impuestas por las bases de datos relacionales, Amazon SimpleDB está optimizado para ofrecer alta disponibilidad y flexibilidad con poca o ninguna carga administrativa. La labor de Amazon SimpleDB pasa inadvertida: se encarga de crear y gestionar varias réplicas de sus datos y las distribuye geográficamente para permitir alta disponibilidad y capacidad de duración. El servicio sólo cobra los recursos realmente consumidos en almacenamiento de los datos y en distribución de las solicitudes. Es posible cambiar el modelo de datos sobre la marcha, y el sistema indexa los datos automáticamente.

Soporte para colas

Amazon Simple Queue Service (Amazon SQS) ofrece un sistema de gestión de colas fiable y ampliable para almacenar mensajes a medida que se transfieren entre sistemas. Mediante Amazon SQS, los desarrolladores pueden transferir datos entre componentes distribuidos de aplicaciones que realizan distintas tareas, sin perder mensajes y sin necesidad de que cada componente esté siempre disponible. Amazon SQS facilita la tarea de creación de un flujo de trabajo automatizado, trabajando en estrecha conexión con Amazon Elastic Compute Cloud (Amazon EC2) y el resto de los servicios web de la infraestructura de AWS.

Amazon SQS funciona utilizando la infraestructura de gestión de mensajes a escala web de Amazon como un servicio web. Cualquier sistema de Internet puede añadir o leer mensajes sin tener instalado ningún software ni configuración de cortafuegos especial. Los componentes de las aplicaciones que utilizan Amazon SQS se pueden ejecutar independientemente y no es necesario que estén en la misma red, que se hayan desarrollado con las mismas tecnologías ni que se ejecuten a la vez.

4.6.1.1.3 Otras consideraciones

Adicionalmente a los servicios previamente comentados, existen otras características configurables y contratables que pueden modificar el comportamiento de los sistemas en la plataforma:

- **Varias ubicaciones**. Amazon EC2 ofrece la posibilidad de colocar instancias en distintas ubicaciones. Las zonas de disponibilidad son regiones diferentes que están diseñadas para estar aisladas de fallos que se produzcan en otras Zonas de disponibilidad, y así poder proteger las aplicaciones de errores en una determinada ubicación.Amazon EC2 está disponible actualmente en nueve regiones: EE.UU. Este (Norte de Virginia), 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 AWS GovCloud (Gobierno USA).
- **Elastic Load Balancing.** Distribuye automáticamente el tráfico entrante de las aplicaciones entre varias instancias de Amazon EC2. Permite conseguir aún más tolerancia a fallos en las aplicaciones, al proporcionar la capacidad de equilibrio de carga necesaria como respuesta al tráfico entrante de aplicaciones. Detecta instancias en mal estado dentro de un conjunto y redirige automáticamente el tráfico hacia las instancias que se encuentran en buen estado, hasta que las primeras se restauran. Se puede habilitar Elastic Load Balancing en una única zona de disponibilidad o en varias para obtener un rendimiento de la aplicación más uniforme.
- **Clústeres de Computación de alto rendimiento (HPC).** Las instancias de informática en clúster, de CPU en clúster y en clúster con memoria elevada se han diseñado específicamente para proporcionar una funcionalidad de red de alto rendimiento. Se pueden iniciar de forma programada en clústeres, lo que permite a las aplicaciones alcanzar el rendimiento de red de baja latencia necesario para la comunicación de nodo a nodo estrechamente asociada. Las instancias en clúster proporcionan también un rendimiento mucho mayor, lo que las hace adecuadas para las aplicaciones personalizadas que necesitan realizar operaciones con un alto consumo de red.
- **Instancias de E/S de alto rendimiento.** Los clientes que requieren un nivel muy alto de volumen de almacenamiento por instancia y un alto nivel de E/S secuenciales para aplicaciones con grandes volúmenes de datos, pueden beneficiarse de esto. Las instancias con alta capacidad de almacenamiento son un tipo de instancia de Amazon EC2 que pueden ofrecer a los clientes un rendimiento secuencial de E/S de 2,4 GB/s y 48 TB de almacenamiento de instancias en 24 unidades de disco duro.
- **VM Import/Export. P**ermite importar imágenes de máquina virtual con facilidad desde el entorno existente a las instancias de Amazon EC2 y volver a exportarlas en el entorno local. Al importar máquinas virtuales listas para utilizar instancias de EC2, se puede aprovechar las inversiones existentes en máquinas virtuales que satisfagan las necesidades del cliente.
- **AWS Marketplace.** AWS Marketplace es una tienda en línea que ayuda a encontrar, comprar e implementar rápidamente el software que se ejecuta en AWS. Se puede utilizar la implementación 1-Click de AWS Marketplace para lanzar rápidamente software preconfigurado.

- **Sistemas Operativos.** Las AMIs se han ido preconfigurado con una lista de sistemas operativos cada vez mayor. Asimismo, se pueden utilizar herramientas de empaquetado para cargar los sistemas operativos propios del cliente. Entre los sistemas operativos que se pueden utilizar actualmente con las instancias de Amazon EC2 se encuentran los siguientes:

Sistemas operativos		
CentOS	Debian	SUSE Linux Enterprise
Amazon Linux	Oracle Enterprise Linux	Ubuntu
Red Hat Enterprise Linux	Windows Server	

Figura 52. Sistemas operativos utilizables en Amazon EC2

- **Software.** AWS Marketplace integra una amplia selección de software comercial y gratuito de proveedores conocidos, diseñado para ejecutarse en instancias de EC2. La Figura 53 muestra algunos de estos productos:

Bases de datos	Pilas de aplicaciones
MongoDB	Lamp
Couchbase Server	Tomcat
SAP HANA One	Ruby on Rails
Riak	Django
Microsoft SQL Server	Node.js

Gestión de contenido	Inteligencia empresarial
WordPress	Informes y análisis de Jaspersoft sobre AWS
Drupal	MicroStrategy
Joomla!	SAP Business Objects
MediaWiki	Kognitio Analytics Platform
Alfresco	

Figura 53. Ejemplos de software utilizable diseñado para Amazaon EC2

4.6.1.1.4 Precios

Amazon establece precios distintos para cada tipo de instancia, sistema operativo y zona geográfica. Para entender mejor la relación de precios, se detallan a continuación los tipos de instancias más habituales y la funcionalidad que aporta cada una de ellas. Posteriormente se reflejarán las tablas de precios que incluirán todos los tipos de instancias admitidos en Amazon EC2.

Instancias estándar

Las instancias estándar de primera generación (M1) proporcionan a los clientes un conjunto equilibrado de recursos y una plataforma de bajo coste adecuada para una amplia diversidad de aplicaciones.

- *Instancia pequeña M1* (predeterminada) de 1,7 GiB de memoria, 1 unidad informática EC2 (1 núcleo virtual con 1 unidad informática EC2), 160 GB de almacenamiento de instancias local, plataforma de 32 o 64 bits.
- *Instancia mediana M1* de 3,75 GiB de memoria, 2 unidades informáticas EC2 (1 núcleo virtual con 2 unidades informáticas EC2), 410 GB de almacenamiento de instancias local, plataforma de 32 o 64 bits.
- *Instancia extragrande M1* de 7,5 GiB de memoria, 4 unidades informáticas EC2 (2 núcleos virtuales con 2 unidades informáticas EC2 cada uno), 850 GB de almacenamiento de instancias local, plataforma de 64 bits.
- *Instancia extragrande M1* de 15 GiB de memoria, 8 unidades informáticas EC2 (4 núcleos virtuales con 2 unidades informáticas EC2 cada uno), 1690 GB de almacenamiento de instancias local, plataforma de 64 bits.

Las instancias estándar de segunda generación (M3) proporcionan a los clientes un conjunto equilibrado de recursos y un nivel mayor de rendimiento del procesamiento en comparación con las instancias estándar de primera generación. Las instancias de este grupo resultan ideales para aplicaciones que requieren un mayor rendimiento absoluto de la CPU y la memoria. Algunas aplicaciones que se beneficiarán del rendimiento de las instancias estándar de segunda generación son la codificación, los sistemas de gestión de contenido de tráfico elevado y memcached.

- *Instancia extragrande M3* de 15 GiB de memoria, 13 unidades informáticas EC2 (4 núcleos virtuales con 3,25 unidades informáticas EC2 cada uno), solo almacenamiento de EBS, plataforma de 64 bits.
- *Instancia extragrande doble M3* con 30 GiB de memoria, 26 unidades informáticas EC2 (8 núcleos virtuales con 3,25 unidades informáticas EC2 cada uno), solo almacenamiento de EBS, plataforma de 64 bits.

Microinstancias

Las microinstancias (t1.micro) ofrecen una pequeña cantidad de recursos de CPU consistentes y permiten ampliar la capacidad de CPU en ráfagas cortas cuando haya nuevos ciclos disponibles. Son adecuadas para aplicaciones con una productividad más baja y sitios web que suelen requerir ciclos de cálculo adicionales con regularidad. Microinstancia con 613 MiB de memoria, hasta 2 ECU (para breves explosiones periódicas), solo almacenamiento de EBS, plataforma de 32 o 64 bits.

Instancias de memoria elevada

Las instancias de esta familia ofrecen una memoria de gran tamaño para aplicaciones de alto rendimiento, incluidas las aplicaciones de almacenamiento en caché y de bases de datos.

- *Instancia extragrande con memoria elevada*: 17,1 GiB de memoria, 6,5 ECU (2 núcleos virtuales con 3,25 unidades informáticas EC2 cada uno), 420 GB de almacenamiento de instancias local, plataforma de 64 bits.
- *Instancia extragrande doble con memoria elevada*: 34,2 GiB de memoria, 13 unidades informáticas EC2 (4 núcleos virtuales con 3,25 unidades informáticas EC2 cada uno), 850 GB de almacenamiento de instancias local, plataforma de 64 bits.
- *Instancia extragrande cuádruple con memoria elevada*: 68,4 GiB de memoria, 26 unidades informáticas EC2 (8 núcleos virtuales con 3,25 unidades informáticas EC2 cada uno), 1690 GB de almacenamiento de instancias local, plataforma de 64 bits.

Instancias de CPU elevada

Las instancias de esta familia tienen, en proporción, más recursos de CPU que memoria (RAM) y resultan adecuadas para aplicaciones que realizan un uso intensivo de la informática.

- *Instancia mediana de CPU elevada:* 1,7 GiB de memoria, 5 unidades informáticas EC2 (2 núcleos virtuales con 2,5 unidades informáticas EC2 cada uno), 350 GB de almacenamiento de instancias local, plataforma de 32 o 64 bits.

Instancia extragrande de CPU elevada: 7 GiB de memoria, 20 unidades informáticas EC2 (8 núcleos virtuales con 2,5 unidades informáticas EC2 cada uno), 1690 GB de almacenamiento de instancias local, plataforma de 64 bits.

Instancias con gran capacidad de almacenamiento

Las instancias de este tipo ofrecen una densidad de almacenamiento por instancia proporcionalmente superior y resultan ideales para aplicaciones que se benefician de un alto rendimiento de E/S secuencial en conjuntos de datos de gran tamaño. Las instancias con gran capacidad de almacenamiento también ofrecen altos niveles de rendimiento de CPU, memoria y red.

- *Instancia extragrande óctuple con gran capacidad de almacenamiento*, 117 GiB de memoria, 35 unidades informáticas de EC2, 24 * 2 TB de almacenamiento de instancias local en la unidad de disco duro, plataforma de 64 bits y Ethernet de 10 Gigabits.

Unidad de sistemas de EC2 (ECU): una unidad de sistemas de EC2 (ECU) proporciona la capacidad de CPU equivalente de un procesador Opteron 2007 o Xeon 2007 de 1,0-1,2 GHz.

Tablas de precios

Una vez descritas las instancias según su funcionalidad, se muestran a continuación las diferentes tablas de precios al combinarlas con los tipos de disponibilidad explicados con anterioridad. Por motivos de simplificación se tomará como base que se trata de un sistema Linux situado en la Unión Europea, ya que los precios varían en función del sistema operativo y la situación geográfica.

- **Instancias según demanda. P**ermiten pagar por la capacidad informática por horas sin compromisos a largo plazo. Esto le liberará de los costes y las complejidades de la planificación, la compra y el mantenimiento del hardware y transformará lo que normalmente son grandes costes fijos en costes variables mucho más reducidos.

Región: UE (Irlanda)

	Uso de Linux/UNIX
Instancias según demanda estándar	
Pequeña (predeterminada)	$0,065 por hora
Mediana	$0,130 por hora
Grande	$0,260 por hora
Extragrande	$0,520 por hora
Instancias según demanda estándar de segunda generación	
Extragrande	$0,550 por hora
Extragrande doble	$1,100 por hora
Microinstancias según demanda	
Micro	$0,020 por hora
Instancias según demanda de memoria elevada	
Extragrande	$0,460 por hora
Extragrande doble	$0,920 por hora
Extragrande cuádruple	$1,840 por hora
Instancias según demanda para CPU de alto rendimiento	
Mediana	$0,165 por hora
Extragrande	$0,660 por hora
Instancias de informática en clúster	
Extragrande óctuple	$2,700 por hora
Instancias según demanda en clúster con memoria elevada	
Extragrande óctuple	$3,750 por hora
Instancias de GPU en clúster	
Extragrande cuádruple	$2,36 por hora
Instancias según demanda con alta capacidad de E/S	
Extragrande cuádruple	$3,410 por hora
Instancias según demanda con gran capacidad de almacenamiento	
Extragrande óctuple	$4,900 por hora

Tabla 9. Precio instancia según demanda Amazon EC2

- **Instancias reservadas.** Las instancias reservadas ofrecen la opción de realizar un pago puntual reducido por cada instancia que desee reservar y recibir a cambio un descuento importante en el cargo por horas de dicha instancia. Existen tres tipos de instancias reservadas (de utilización ligera, media e intensa) que permiten equilibrar el importe que paga por anticipado con su precio por hora efectivo. Por simplicidad se muestra únicamente la tabla de precios para las instancias de utilización media que suponen valores intermedios entre las otras dos opciones existentes.

Región: UE (Irlanda)

	1 año de plazo		3 años de plazo	
	Pago anticipado	**Tarifa por hora**	**Pago anticipado**	**Tarifa por hora**
Instancias reservadas estándar				
Pequeña (predeterminada)	$139	$0,027 por hora	$215	$0,022 por hora
Mediana	$277	$0,054 por hora	$430	$0,043 por hora
Grande	$554	$0,108 por hora	$860	$0,087 por hora
Extragrande	$1108	$0,215 por hora	$1720	$0,173 por hora
Instancias reservadas estándar de segunda generación				
Extragrande	$1217	$0,236 por hora	$1919	$0,19 por hora
Extragrande doble	$2434	$0,472 por hora	$3838	$0,38 por hora
Microinstancias reservadas				
Micro	$54	$0,01 por hora	$82	$0,01 por hora
Instancias reservadas de memoria elevada				
Extragrande	$651	$0,13 por hora	$992	$0,104 por hora
Extragrande doble	$1302	$0,26 por hora	$1984	$0,208 por hora
Extragrande cuádruple	$2604	$0,52 por hora	$3968	$0,416 por hora
Instancias reservadas para CPU de alto rendimiento				
Mediana	$370	$0,072 por hora	$571	$0,063 por hora
Extragrande	$1480	$0,288 por hora	$2284	$0,252 por hora
Instancias reservadas de informática en clúster				
Extragrande cuádruple	N/A	N/A	N/A	N/A
Extragrande óctuple	$4146	$0,75 por hora	$6378	$0,75 por hora
Instancias reservadas en clúster con memoria elevada				
Extragrande óctuple	$5958	$1,152 por hora	$9006	$1,014 por hora
Instancias reservadas de GPU en clúster				
Extragrande cuádruple	N/A	N/A	N/A	N/A
Instancias reservadas con alta capacidad de E/S				
Extragrande cuádruple	$5973	$1,379 por hora	$9133	$1,022 por hora
Instancias reservadas con gran capacidad de almacenamiento				
Extragrande óctuple	$9200	$1,809 por hora	$14103	$1,581 por hora

Tabla 10. Precio instancia reservada de utilización media en Amazon EC2

- **Instancias puntuales.** Las instancias puntuales permiten realizar ofertas por capacidad de Amazon EC2 que no haya sido utilizada. Las instancias se cobran según el precio puntual, que fija Amazon EC2 y que fluctúa de forma periódica dependiendo de la oferta y de la demanda para la capacidad de instancias puntuales. Para utilizar las instancias puntuales, se debe realizar una solicitud de instancia puntual, el tipo de instancia, la zona de disponibilidad deseada, el número de instancias puntuales que se quiere ejecutar y el precio máximo que se está dispuesto a pagar por hora de instancia.

Región: UE (Irlanda)

	Uso de Linux/UNIX	Uso de Windows
Instancias puntuales estándar		
Pequeña (predeterminada)	$0,016 por hora	$0,032 por hora
Mediana	$0,032 por hora	$0,064 por hora
Grande	$0,064 por hora	$0,128 por hora
Extragrande	$0,128 por hora	$0,256 por hora
Instancias puntuales estándar de segunda generación		
Extragrande	$0,14 por hora	$0,268 por hora
Extragrande doble	$0,28 por hora	$0,536 por hora
Microinstancias puntuales		
Micro	$0,006 por hora	$0,011 por hora
Instancias puntuales de memoria elevada		
Extragrande	$0,094 por hora	$0,158 por hora
Extragrande doble	$0,189 por hora	$0,317 por hora
Extragrande cuádruple	$0,378 por hora	$0,634 por hora
Instancias puntuales para CPU de alto rendimiento		
Mediana	$0,044 por hora	$0,096 por hora
Extragrande	$0,176 por hora	$0,384 por hora
Instancias de informática en clúster		
Extragrande cuádruple	N/A*	N/A*
Extragrande óctuple	$0,488 por hora	N/A*
Instancias puntuales en clúster con memoria elevada		
Extragrande óctuple	$0,343 por hora	N/A*
Instancias de GPU en clúster		
Extragrande cuádruple	$0,54 por hora	N/A*

Tabla 11. Precio instancia puntual en Amazon EC2

Existen otras tablas de precios asociadas al resto de servicios, como CloudWatch, Elastic Load Balancing, etc., pero para el objetivo del que se ocupa este apartado la información resulta suficiente para representar unos valores económicos aproximados de los aspectos de contratación más comunes.

4.6.1.2 Windows Azure

Windows Azure es una plataforma de nube abierta y flexible que permite compilar, implementar y administrar aplicaciones rápidamente en una red global de centros de datos administrados por Microsoft. Puede compilar aplicaciones en cualquier lenguaje, herramienta o marco, permitiendo además integrar sus aplicaciones de nube públicas con el entorno de TI existente.

Es una plataforma que permite desarrollar en diversos lenguajes así como la comunicación con cualquier entorno externo. Además, si bien Visual Studio es la herramienta más productiva a la hora de trabajar con la plataforma, se dispone de herramientas y SDKs5 para otros sistemas y entornos.

4.6.1.2.1 Características principales

A continuación se presentan las principales características que definen esta infraestructura de computación en nube:

- **Siempre Disponible.** Disponibilidad 24x7x365. Con "caching" de Windows Azure se puede mantener la alta disponibilidad para los objetos en caché. No es necesario cambiar ningún código, solo se recalcula la capacidad de memoria necesaria para la carga de trabajo. Con la alta disponibilidad, los objetos en caché se replican dentro de la misma implementación de servicios en la nube para ofrecer resistencia contra los errores de hardware. Las copias secundarias también se colocan entre distintos dominios de errores y de actualización para aumentar la disponibilidad. Si, por cualquier razón, se produce un error en alguna de las máquinas virtuales del clúster de caché, este puede usar las copias secundarias para evitar la pérdida de datos.

- **Alto nivel de servicio.** Windows Azure se entrega un SLA mensual del 99,95 % que permite compilar y ejecutar aplicaciones de alta disponibilidad sin importar la infraestructura. Proporciona revisiones automáticas del sistema operativo y de los servicios, equilibrio de carga de red integrado y resistencia ante errores de hardware. Admite un modelo de implementación con el que se puede actualizar una aplicación sin inactividad (downtime).

- **Abierto.** Windows Azure permite utilizar cualquier lenguaje, marco o herramienta para crear aplicaciones. Las características y los servicios se exponen utilizando protocolos REST abiertos. Las bibliotecas de cliente de Windows Azure están disponibles para varios lenguajes de programación, se comercializan bajo una licencia de código abierto y se hospedan en GitHub.

- **Servidores ilimitados, almacenamiento ilimitado.** Windows Azure permite escalar aplicaciones a cualquier tamaño con facilidad. Es una plataforma de autoservicio totalmente automatizada que permite el aprovisionamiento de recursos en cuestión de minutos. El uso de recursos aumenta o disminuye de manera flexible en función de

las necesidades. Solo se pagan los recursos que usa la aplicación. Windows Azure está disponible en varios centros de datos del mundo, lo que permite implementar las aplicaciones cerca de los clientes.

- **Gran capacidad.** Windows Azure proporciona una plataforma flexible en la nube que puede satisfacer los requisitos de cualquier aplicación. Permite hospedar y ampliar el código de aplicación dentro de roles de proceso de un modo totalmente confiable. Los datos se pueden almacenar en bases de datos SQL relacionales, almacenes de tablas NoSQL y almacenes de blobs no estructurados, y existe la opción de usar la funcionalidad de Hadoop e inteligencia empresarial para la minería de datos. Se puede aprovechar la sólida funcionalidad de mensajería de Windows Azure para habilitar aplicaciones distribuidas escalables, así como para entregar soluciones híbridas que se ejecuten en la nube y en un entorno empresarial local.

- **Cache distribuida y CDN (Content Distribution Network).** Los servicios de caché distribuida y red de entrega de contenido (CDN) de Windows Azure permiten reducir la latencia y ofrecer aplicaciones con un gran rendimiento en cualquier lugar del mundo.

4.6.1.2.2 Servicios

Como en el caso anterior, se presenta una descripción de los servicios ofrecidos por Windows Azure.

Autoescabilidad

Se pueden escalar automáticamente las aplicaciones de Windows Azure basándolas en reglas definidas específicamente. Estas reglas pueden ayudar a Windows Azure a mantener su rendimiento en respuesta a los cambios en la carga de trabajo, y al mismo tiempo controlar los costes asociados con el alojamiento de la aplicación en Windows Azure. Junto con la escalabilidad, aumentando o disminuyendo el número de instancias de rol de la aplicación, el bloque también permite utilizar otras medidas de escalabilidad tales como funcionalidades determinadas de estrangulamiento ("throttling") dentro de la aplicación o el uso de las acciones definidas por el usuario.

Brinda además la posibilidad de optar por implementar el bloque en un rol de Windows Azure o en una aplicación interna. El Bloque de Aplicación Autoescalable forma parte del Pack de la Enterprise Library 5.0 de Microsoft Integration para Windows Azure. El bloque utiliza dos tipos de reglas para definir el comportamiento automático de escalabilidad para su aplicación:

- *Reglas de restricciones*: Para establecer los límites superior e inferior en el número de instancias, por ejemplo, digamos que 8:00-10:00 todos los días se quiere un mínimo de cuatro y un máximo de seis instancias, se utiliza una regla de restricción.

- *Reglas Reactivas*: Para permitir que el número de instancias de rol puedan cambiar en respuesta a los cambios impredecibles en la demanda, se utilizan reglas reactivas.

Blueprints / Imágenes para acelerar el aprovisionamiento

Las máquinas virtuales entregadas a demanda ofrecen una infraestructura de computación escalable cuando se necesita aprovisionar rápidamente recursos para satisfacer las necesidades de un negocio en crecimiento. Es posible obtener máquinas virtuales de los sistemas operativos Linux y Windows Server en múltiples configuraciones.

Los blueprints de Windows Azure ofrecen la posibilidad de desbloquear la cartera de TI y la infraestructura de suministro al ritmo que el negocio lo requiere. Para ello, simplemente hay que elegir la configuración deseada (instancias de memoria estándar o alta) y seleccionar una imagen de la galería de imágenes de máquinas virtuales.

Las Máquinas virtuales de Windows Azure ofrecen a los sistemas y aplicaciones la posibilidad de mover los discos duros virtuales (VHD) desde las instalaciones locales a la nube (y viceversa).

Soporte para almacenamiento de datos

En este apartado se provee información acerca de las alternativas ofrecidas por la plataforma Windows Azure para la persistencia de datos. Se analizarán las siguientes alternativas: SQL Server en máquinas virtuales de Windows Azure, Base de datos SQL, Almacenes de tablas NoSQL, Blob no estructurado.

SQL Server en máquinas virtuales de Windows Azure

Cuando las aplicaciones requieren funcionalidad completa de SQL Server, las máquinas virtuales es la solución ideal. La ejecución de SQL Server en máquinas virtuales es una solución adecuada en los escenarios siguientes:

- Para desarrollar y probar nuevas aplicaciones de SQL Server rápidamente. No es necesario esperar semanas para el aprovisionamiento local de hardware, sino que basta con captar la imagen de SQL Server correcta en la galería de imágenes.

- Para hospedar aplicaciones de SQL Server de nivel 2 y nivel 3 existentes. Gracias a los distintos tamaños de VM entre los que elegir y dada la compatibilidad total con SQL Server, es posible trasladar las aplicaciones de SQL Server locales existentes y disfrutar de la eficacia de la computación en la nube.

- Para realizar copias de seguridad y restauraciones de bases de datos locales. Es posible realizar la copia de seguridad de una base de datos local en un almacenamiento en blobs de Windows Azure, y poder así restaurar la base de datos en una máquina virtual de Windows Azure en caso de que sea necesaria la recuperación ante desastres en el entorno local.

- Para ampliar aplicaciones locales. Se pueden crear aplicaciones híbridas que utilicen activos locales y máquinas virtuales de Windows Azure para disfrutar de una mayor eficacia y alcance global.
- Para crear aplicaciones de varias capas en la nube. Se puede crear una aplicación de varias capas que utilice la funcionalidad de escalado única del servicio Base de datos SQL para la capa de aplicación, y que aproveche la compatibilidad completa de SQL Server en Máquinas virtuales de Windows Azure para la capa de base de datos.

Base de datos SQL

Para aquellas aplicaciones que necesitan una base de datos relacional completamente funcional como servicio, Windows Azure ofrece la base de datos SQL, antes denominada Base de datos de SQL Azure. La base de datos SQL ofrece un alto nivel de interoperabilidad, lo que permite a los clientes crear aplicaciones en la mayoría de los principales marcos de desarrollo. Además, la base de datos SQL, basada en las tecnologías probadas de SQL Server, permite utilizar los conocimientos y la experiencia existente para reducir el tiempo de solución, así como crear o ampliar aplicaciones entre los sistemas locales y la nube.

Tablas

Las tablas ofrecen funcionalidad NoSQL para las aplicaciones que requieren el almacenamiento de grandes cantidades de datos no estructurados. Las tablas son un servicio administrado con certificación ISO 27001 que se pueden escalar automáticamente para satisfacer un rendimiento y volumen masivos de hasta 100 terabytes, accesibles prácticamente desde cualquier lugar a través de REST y las API administradas.

Blob no estructurado

Los blobs son el modo más sencillo de almacenar grandes cantidades de texto no estructurado o datos binarios tales como vídeo, audio e imágenes. Los blobs son un servicio administrado con certificación ISO 27001 que se pueden escalar automáticamente para satisfacer un rendimiento y volumen masivos de hasta 100 terabytes, accesibles prácticamente desde cualquier lugar a través de REST y las API administradas.

Soporte para colas

El bus de servicio de Colas soporta un modelo de comunicación de mensajería negociado. Cuando se utilizan colas, los componentes de una aplicación distribuida no se comunican directamente entre sí, sin embargo, intercambian mensajes a través de una cola, la cual actúa como un intermediario. Un productor de mensajes (remitente) envía un mensaje a la cola y continúa su procesamiento. Asincrónicamente, un consumidor de mensajes (receptor) obtiene el mensaje de la cola y lo procesa. El productor no tiene que esperar una respuesta por parte del consumidor para poder continuar el proceso y enviar más mensajes. Las colas ofrecen la implementación de la técnica "primero que entra, primero que sale" (modelo conocido por sus siglas en inglés como "FIFO") enviando mensajes a uno o más consumidores que compiten por el tratamiento del mensaje. Es decir, los mensajes suelen ser recibidos y procesador por los receptores en el orden en que fueron añadidos a la cola, y cada mensaje es recibido y procesado por un único consumidor.

4.6.1.2.3 Precios

Proceso (Máquinas Virtuales)

Existen distintos combinaciones de precios, dependiendo del sistema operativo (Windows/ Linux/SQL Server/Servidor BizTalk) y la modalidad de pago (pago por uso/plan de 6 o 12 meses). Para ilustrar las tablas de precios se seleccionará un sistema Linux y el plan de pago por uso.

El precio de máquinas virtuales se calcula por minuto. Los precios figuran por tarifa horaria y se facturan como número total de minutos cuando las máquinas virtuales se ejecutan en fracciones de hora. Los precios para Windows incluyen el costo de licencia de Windows Server.

Las *instancias estándar* proporcionan un conjunto óptimo de recursos de proceso, memoria y entrada y salida para ejecutar una amplia gama de aplicaciones.

NOMBRE DEINSTANCIA DE PROCESO	NÚCLEOS VIRTUALES	RAM	PRECIO POR HORA
Extra pequeña (A0)	Uso compartido	768 MB	**€0,0149** (~€11,18/mes)
Pequeña (A1)	1	1,75 GB	**€0,0447** (~€33,52/mes)
Mediana (A2)	2	3,5 GB	**€0,0894** (~€66,28/mes)
Grande (A3)	4	7 GB	**€0,1788** (~€133,31/mes)
Extra grande (A4)	8	14 GB	**€0,3575** (~€265,86/mes)

* Basado en 744 horas al mes

Tabla 12. Precio instancia estándar de máquina virtual en Windows Azure

Las *instancias de memoria intensiva* proporcionan más cantidad de memoria, óptima para ejecutar aplicaciones de alto rendimiento como las bases de datos.

NOMBRE DEINSTANCIA DE PROCESO	NÚCLEOS VIRTUALES	RAM	PRECIO POR HORA
A6	4	28 GB	€0,7596 (~€565,23/mes)
A7	8	56 GB	€1,5192 (~€1.130,46/mes)

* Basado en 744 horas al mes

Tabla 13. Precio instancia de memoria intensiva de máquina virtual en Windows Azure

Proceso (Sitios Web)

"Sitios web" de Windows Azure permite implementar aplicaciones web en una infraestructura en la nube escalable y confiable. Desarrolladores y propietarios de sitios web pueden escalar vertical y horizontalmente de manera rápida para satisfacer sus necesidades de tráfico y aplicación. Existen las modalidades de "Sitios web" y "Conexiones SSL", y partimos del plan de pago por uso para ilustrar las tablas de precios. El producto se ofrece en tres niveles: Gratis, Compartido (vista previa) y Estándar.

El nivel estándar ofrece varios tamaños de instancia, así como la posibilidad del escalado en función de los cambios en los requisitos de capacidad. Los precios del nivel estándar son los que se muestran en la Tabla 14:

TAMAÑO	NÚCLEOS DE CPU	MEMORIA	PRECIO POR HORA
Pequeña	1	1,75 GB	€0,075 (~€56/mes)*
Mediana	2	3,5 GB	€0,149 (~€111/mes)*
Grande	4	7 GB	€0,298 (~€222/mes)*

* Basado en 744 horas al mes

Tabla 14. Precios de nivel estándar de "Sitio Web" en Windows Azure

En la siguiente tabla se muestra una comparación entre las características de los diferentes niveles que se ofrecen (Tabla 15):

	GRATIS	USO COMPARTIDO (VISTA PREVIA)	ESTÁNDAR
CPU	Uso compartido[1]	Uso compartido[1]	Dedicado
Compatibilidad con dominios personalizados:	No disponible	Disponible	Disponible
SSL de dominios personalizados	No disponible	No disponible	Consulte los precios de SSL
Escalado	No disponible	Hasta 6 instancias	Hasta 10 instancias
Sitios[2]	10	100	500
Almacenamiento[2]	1 GB	1 GB	10 GB
Base de datos relacional[3] (opcional)	20 MB incluidos Se aplican las tarifas estándar para más capacidad	20 MB incluidos Se aplican las tarifas estándar para más capacidad	20 MB incluidos Se aplican las tarifas estándar para más capacidad
Transferencia de datos de salida[2]	Hasta 165 MB al día	Se aplican las tarifas estándar	Se aplican las tarifas estándar

[1] Los niveles gratis y de uso compartido (vista previa) incluyen 60 minutos y 240 minutos de capacidad de CPU al día, respectivamente.

[2] Estas cuotas se aplican por subregión, a menos que se especifique lo contrario.

[3] Hay una base de datos SQL de Windows Azure de 20 MB y una base de datos MySQL de 20 MB disponibles en el nivel de suscripción durante los doce primeros meses de uso; a partir de ese momento, se aplican las tarifas estándar.

Tabla 15. Comparativa entre los niveles de "Sitios Web" en Windows Azure

Proceso (Servicios en la nube)

Servicios en la nube de Windows Azure evita la necesidad de administrar la infraestructura de servidor. Con los roles de web y trabajo, permite compilar, implementar y administrar aplicaciones modernas rápidamente. Se partirá del plan de pago por uso para ilustrar las tablas de precios.

Las *instancias estándar* proporcionan un conjunto óptimo de recursos de proceso, memoria y entrada y salida para ejecutar una amplia gama de aplicaciones.

NOMBRE	NÚCLEOS VIRTUALES	RAM	PRECIO POR HORA
Extra pequeña (A0)	Uso compartido	768 MB	**€0,0149** (~€11,18/mes)
Pequeña (A1)	1	1,75 GB	**€0,0596** (~€44,69/mes)
Mediana (A2)	2	3,5 GB	**€0,1192** (~€88,62/mes)
Grande (A3)	4	7 GB	**€0,2384** (~€177,24/mes)
Extra grande (A4)	8	14 GB	**€0,4767** (~€354,48/mes)

* Basado en 744 horas al mes

Tabla 16. Precio instancia de estándar de servicios en la nube en Windows Azure

Las *instancias de memoria intensiva* proporcionan una gran cantidad de memoria, óptima para ejecutar aplicaciones de alto rendimiento, como las bases de datos. La memoria se asigna asimétricamente por núcleo virtual, a 7 GB por núcleo virtual.

NOMBRE	NÚCLEOS VIRTUALES	RAM	PRECIO POR HORA
A6	4	28 GB	€0,6703 (~€498,95/mes)
A7	8	56 GB	€1,3405 (~€997,16/mes)

* Basado en 744 horas al mes

Tabla 17. Precio instancia de memoria intensiva de servicios en la nube en Windows Azure

Servicio de Datos (Almacenamiento)

Windows Azure Storage está diseñado para almacenar y recuperar grandes volúmenes de datos de una manera rentable, con facilidad de acceso y durabilidad. Ofrece almacenamiento de datos no relacionales en blobs, tablas, colas y unidades.

Se parte del plan de pago por uso para ilustrar las tablas de precios. El almacenamiento se cobra en función del volumen de almacenamiento (la cantidad de datos almacenados en blobs, colas y unidades) y en función de las transacciones de almacenamiento (número de operaciones de lectura y escritura en el almacenamiento).

Se ofrecen dos versiones de almacenamiento: con redundancia geográfica y con redundancia local.

- *Almacenamiento con redundancia local*: se mantienen varias réplicas de los datos de una misma subregión a fin de proporcionar una alta durabilidad.
- *Almacenamiento con redundancia geográfica*: representa una durabilidad de datos adicional, ya que los datos se replican entre dos subregiones situadas a cientos de kilómetros de distancia, dentro de la misma región. En ambas subregiones se mantienen varias réplicas de los datos.

CAPACIDAD DE ALMACENAMIENTO	CON REDUNDANCIA GEOGRÁFICA	CON REDUNDANCIA LOCAL
Primer 1 TB[1]/mes	€0,0708 por GB	€0,0522 por GB
Siguientes 49 TB / mes	€0,0596 por GB	€0,0485 por GB
Siguientes 450 TB / mes	€0,0522 por GB	€0,0447 por GB
Siguientes 500 TB / mes	€0,0485 por GB	€0,041 por GB
Siguientes 4.000 TB/mes	€0,0447 por GB	€0,0336 por GB
Siguientes 4.000 TB/mes	€0,041 por GB	€0,0276 por GB
Más de 9.000 TB / mes	Ponerse en contacto con nosotros	Ponerse en contacto con nosotros

* Basado en 744 horas al mes

[1] 1 TB = 1.024 GB

Transacciones de almacenamiento

€0,0075 por 100.000 transacciones

Las transacciones incluyen las operaciones de lectura y escritura en el almacenamiento.

Tabla 18. Precio de capacidad de almacenamiento en Windows Azure

Servicio de Datos (Base de Datos SQL)

Base de datos SQL de Windows Azure (antes SQL Azure) es un servicio de base de datos relacional muy completo y totalmente administrado que ofrece una experiencia de alta productividad, incorpora tecnología demostrada de SQL Server y ofrece funcionalidad de clase empresarial. Permite compilar, ampliar y escalar las aplicaciones relacionales rápidamente en la nube, con herramientas conocidas y con la eficacia de la tecnología Microsoft SQL Server.

Base de datos SQL está disponible en las siguientes ediciones:

- Web y Business: Las bases de datos relacionales que ofrecen las ediciones Web y Business se ejecutan en recursos compartidos y tienen réplicas integradas en un centro de datos. Web Edition admite bases de datos de hasta 5 GB y Business Edition admite bases de datos de hasta 150 GB. Ambas ediciones admiten el escalado dinámico de miles de bases de datos distribuidas. Estas ediciones se cobran en función del volumen real de la base de datos, que se mide en GB.

- Premium (vista previa): La edición Premium, creada sobre la misma base que las ediciones Web y Business, ofrecen acceso a funcionalidad de clase empresarial fácil de ampliar o reducir en función de las necesidades. Lo especial de la edición Premium es la posibilidad de reservar capacidad. La reserva garantiza una cantidad fija de capacidad que no se comparte con ninguna otra base de datos. De este modo, el rendimiento es mayor. Premium está actualmente en versión de vista previa. Se cobra en función del tamaño de reserva asignado a la base de datos y el volumen de almacenamiento de la base de datos.

Para la ilustración de las tablas de precios, se parte desde el plan de pago por uso. Esta información se muestra en la Tabla 19:

TAMAÑO DE BASE DE DATOS	PRECIO POR BASE DE DATOS AL MES (CON PRORRATEO DIARIO)	
0-100 MB	€3,7198	
100 MB - 1 GB	€7,4396	
1 GB - 10 GB	€7,4396 por el primer GB	€2,9759 por cada GB adicional
10 GB - 50 GB	€34,23 por los 10 primeros GB	€1,4865 por cada GB adicional
50 GB - 150 GB	€93,74 por los 50 primeros GB	€0,744 por cada GB adicional

Tabla 19. Precio de bases de datos SQL en Windows Azure

Servicio de Datos (Backup)

Windows Azure Backup administra copias de seguridad en la nube con herramientas similares a las de Windows Server 2012, Windows Server 2012 Essentials o System Center 2012 Data Protection Manager.

Para la representación de la tabla de precios de los servicios de backup se patirá del plan de pago por uso. La copia de seguridad se facturará según la cantidad de datos almacenados en el servicio Backup. No habrá ningún costo adicional por el ancho de banda, almacenamiento, transacciones de almacenamiento o proceso ni otros recursos asociados con el servicio Backup.

DATOS COMPRIMIDOS ALMACENADOS AL MES	PRECIO (VISTA PREVIA)
Primeros 5 GB / mes [1]	Gratis
Más de 5 GB / mes	€0,1862 por GB al mes

[1] Para las ofertas de crédito monetario, los servicios de backup se cobrarán en el nivel facturable.

Tabla 20. Precio de servicio de backup en Windows Azure

4.6.1.2.4 Soporte técnico

Windows Azure proporciona opciones de soporte técnico flexibles para clientes de todos los tamaños, desde desarrolladores que inician su actividad en la nube a empresas que implementan aplicaciones críticas para el negocio. Las opciones que se ofrecen se presentan en la Tabla 21:

	INCLUIDO	DEVELOPER	STANDARD	PROFESSIONAL DIRECT	PREMIER
		€21,60	€223,41	€744,70	
Facturación y administración de suscripciones	✓	✓	✓	✓	✓
Foros de la comunidad	✓	✓	✓	✓	✓
Panel de servicios	✓	✓	✓	✓	✓
Envío de incidentes web		✓	✓	✓	✓
Mantenimiento (24x7)		✓	✓	✓	✓
Tiempo de respuesta más rápido		<8 horas	<2 horas	<1 hora	<15 min
Asistencia telefónica (devolución de llamada)			3/mes	Ilimitado	Ilimitado
Administración de entrega de servicio				En grupo	Asignado
Tratamiento de prioridades				✓	✓
Línea telefónica para remisión a instancia superior				✓	✓
Asesoramiento				Limitado	Completo
Servicios in situ					✓
Aprendizaje para desarrolladores					✓

Tabla 21. Oferta de soporte técnico de Windows Azure

Como ocurría en el caso de la plataforma anterior, existe un mayor número de servicios adicionales y tablas de precios, sin embargo, las opciones presentadas ofrecen información suficiente para poder realizar una comparativa entre las compañías y seleccionar la que mejor se adapte a las necesidades de la investigación.

4.6.1.3 Google App Engine

Google App Engine permite crear y alojar aplicaciones web en los mismos sistemas escalables con los que funcionan las aplicaciones de Google. Ofrece procesos de desarrollo e implementación rápidos, y una administración sencilla, sin necesidad de preocuparse por el hardware, las revisiones o las copias de seguridad.

Las aplicaciones Google App Engine son fáciles de crear, de mantener y de escalar a medida que el tráfico y las necesidades de almacenamiento de datos crecen. Con App Engine no es necesario mantener ningún servidor. Basta con cargar la aplicación, y ésta ya se encontrará lista para servir a los usuarios.

4.6.1.3.1 Características Principales

En esta sección se describen las principales características de la plataforma Google App Engine obtenidas desde su sitio oficial:

- **Rápido y sin riesgos.** Crear una aplicación en App Engine no sólo resulta fácil, sino que, además, es gratis. Se puedes crear una cuenta y publicar una aplicación que se podrá utilizar inmediatamente sin ningún coste ni obligación. Una aplicación de una cuenta gratuita dispone de hasta 1 GB de espacio y admite hasta cinco millones de vistas mensuales. Cuando se esté listo para más, se puede habilitar la facturación, configurar un presupuesto diario máximo y asignarle el presupuesto a cada recurso en función de las necesidades.

- **Fácil de usar.** Google App Engine incluye las herramientas necesarias para crear, probar, ejecutar y actualizar aplicaciones. Es tan fácil como crear el código de la aplicación, probarla localmente y subirla a Google únicamente haciendo clic en un botón o introduciendo una secuencia en el símbolo del sistema. Una vez que se haya subido la aplicación a Google, ellos se encargan de alojarla y de escalarla. No hay preocupación de la administración del sistema, de la activación de instancias nuevas de la aplicación, de la fragmentación de la base de datos ni de la adquisición de equipos. Google App Engine se ocupa del mantenimiento.

- **Conjunto rico de APIs.** Google App Engine provee un conjunto rico de APIs de servicios de fácil uso, permitiendo crear servicios rápidamente.

- **Escabilidad Inmediata.** Las aplicaciones podrán aprovechar las mismas tecnologías escalables sobre las que están creadas las aplicaciones de Google como, por ejemplo, BigTable y GFS. App Engine dispone de una función de escalabilidad automática, así que lo único que hay que hacer es crear el código de la aplicación, del resto se encarga Google independientemente del número de usuarios y de la cantidad de datos que se requiera.

- **Infraestructura probada.** La infraestructura de Google es famosa por su gran fiabilidad y por su alto rendimiento. Con App Engine, los clientes pueden aprovechar los diez años de experiencia que posee Google en la ejecución de sistemas escalables de forma masiva y concebidos para el rendimiento. A todas las aplicaciones App Engine se les aplica las mismas políticas de seguridad, privacidad y protección de datos que a las demás aplicaciones de Google.

4.6.1.3.2 Servicios

Al igual que ocurría en las plataformas anteriores, se presenta a continuación los servicios más relevantes ofertados por Google App Engine.

Autoescabilidad

Google App Engine está diseñado para alojar aplicaciones con muchos usuarios simultáneos. Cuando una aplicación puede servir a muchos usuarios simultáneos sin degradar su rendimiento, se dice que es escalable.

Las aplicaciones escritas para App Engine escalan automáticamente. A medida que aumenta la demanda de la aplicación, App Engine asigna un mayor número de recursos y administra su utilización. La aplicación en sí no necesita saber nada acerca de los recursos que utiliza.

Soporte para almacenamiento de datos

El almacenamiento de datos de una aplicación web escalable puede llegar a ser complicado. Un usuario puede interactuar con cualquiera de las docenas de servidores web en un momento dado y la siguiente petición podría ir a un servidor diferente de la anterior solicitud. Todos los servidores web deben interactuar con los datos, que también se extienden a través de docenas de máquinas que posiblemente se encuentran en diferentes lugares del mundo.

Con Google App Engine no es necesario preocuparse por estos aspectos. La infraestructura de App Engine se encarga de toda la distribución, la replicación y el equilibrio de carga de los datos detrás de una sencilla API, y se obtiene un motor de búsqueda de gran alcance, garantizando también las transacciones.

El repositorio de datos de App Engine y el almacén de datos de alta replicación (HRD) utilizan el algoritmo Paxos para replicar datos a través de múltiples centros de datos. Los datos se escriben en el almacén de datos en objetos conocidos como entidades. Cada entidad tiene una clave que identifica de manera única. Una entidad puede designar opcionalmente otra entidad como su matriz, la primera entidad es un hijo de la entidad matriz. Las entidades en el almacén de datos forman así un espacio de estructura jerárquica similar a la estructura de directorios de un sistema de archivos.

El almacén de datos es extremadamente resistente frente a fallos catastróficos, sin embargo, garantizar la consistencia puede ser bastante diferente a lo que suele ser habitual. Las entidades que descienden de un antepasado común se dice que pertenecen a un mismo grupo de entidades, las llaves de su ancestro común es la clave principal del grupo, que sirve para identificar a todo el grupo. Las consultas en una única entidad de su grupo, llamado consultas de antepasados, hacen referencia a la clave principal en lugar de la clave de una entidad específica. Los grupos de entidad son una unidad de consistencia y transaccionalidad: mientras que las consultas sobre múltiples grupos de entidades pueden devolver resultados viciados, eventualmente consistentes, aquellas limitadas a un único grupo de entidad siempre retornan actualizadas, produciendo resultados muy consistentes.

Soporte para colas

Una aplicación puede realizar tareas además de responder a solicitudes web. Las aplicaciones implementadas en Google App Engine pueden ejecutar estas tareas siguiendo la programación que se configure, por ejemplo, cada día o cada hora. Asimismo, es posible ejecutar tareas que ella misma ha añadido a una cola, como una tarea en segundo plano creada durante el procesamiento de una solicitud. Las tareas programadas también se conocen como "tareas cron", administradas por el servicio cron.

Las colas de tareas se incluyen actualmente como una función experimental. En este momento, solo el entorno de tiempo de ejecución Python puede utilizar colas de tareas. Se incluirá una interfaz de cola de tareas para aplicaciones Java en poco tiempo.

Con el API de la cola de tareas, las aplicaciones pueden realizar fuera de solicitudes de usuario trabajos que se han iniciado dentro de ellas. Si una aplicación necesita ejecutar algún trabajo en segundo plano, puede utilizar el API de la cola de tareas para organizarlo en pequeñas unidades discretas llamadas tareas. A continuación, la aplicación inserta estas tareas en una o más colas. App Engine detecta automáticamente nuevas tareas y las ejecuta cuando los recursos del sistema lo permiten.

4.6.1.3.3 Precios

Cada aplicación de App Engine puede consumir una cantidad fija de los recursos informáticos de forma gratuita, que se define por un conjunto de cuotas. Si la aplicación necesita más recursos se puede hacer una aplicación de pago, permitiendo la facturación y la vinculación a una tarjeta de crédito o cuenta bancaria para el pago automático. Cuando la aplicación utiliza los recursos más allá de los cupos libres, se le cobrará sólo para el uso adicional, hasta un importe máximo diario que usted especifique. Los cargos se acumulan mensualmente y se pagan al comienzo del mes siguiente.

Google App Engine proporciona un cuadro resumen con los precios de sus servicios más significativos. Esta tabla resumen se corresponde con la Tabla 22 mostrada a continuación:

Resource	Unit	Unit cost
Outgoing Bandwidth	Gigabytes	0,09€
Frontend Instances (F1 class)	Instance hours	0,06€
Frontend Instances (F2 class)	Instance hours	0,12€
Frontend Instances (F4 class)	Instance hours	0,24€
Frontend Instances (F4_1G class)	Instance hours	0,36€
Discounted Instances	Instance hours	0,04€
Backend Instances (B1 class)	Hourly per instance	0,06€
Backend Instances (B2 class)	Hourly per instance	0,12€
Backend Instances (B4 class)	Hourly per instance	0,24€
Backend Instances (B4_1G class)	Hourly per instance	0,36€
Backend Instances (B8 class)	Hourly per instance	0,47€
Stored Data (Blobstore)	Gigabytes per month	0,10€
Stored Data (Datastore)	Gigabytes per month	0,13€
Stored Data (Logs Data)	Gigabytes per month	0,18€
Stored Data (Task Queue)	Gigabytes per month	0,18€
Dedicated Memcache	Gigabytes per hour	0,09€
Logs API	Gigabytes	0,09€
SNI SSL certificates	Sets of five SNI certificate slots per month	6,67€
SSL Virtual IPs (VIPs)	Virtual IP per month	28,89€
PageSpeed bandwidth	Gigabytes (in addition to outgoing bandwidth charges)	0,29€

Tabla 22. Tarifas de facturación de recursos de Google App Engine

Esta tabla resumen muestra de manera concentrada los precios aplicables a los servicios de facturación más comunes de este tipo de plataformas y permite aproximar de manera sencilla el coste asociado a la contratación de los servicios en la nube de Google, sirviendo de base para establecer comparaciones con otras infraestructuras de la misma índole.

4.6.2 Discusión

Tras realizar un estudio entre las diferentes ofertas planteadas, se puede comprobar cómo en la mayoría de los aspectos las diferencias existentes son mínimas. Todas ellas ofrecen los servicios propios del paradigma como son el autoescalado, gestión de copias de seguridad, servicios de almacenamiento, mantenimiento, etc.

Las diferencias en cuanto a precios tampoco son significativas, ya que todos ellos disminuyen a medida que el número de instancias o servicios contratados aumenta, sin embargo, el sistema desarrollado no precisa inicialmente de grandes cantidades de instancias o almacenamiento. En este aspecto Google App Engine proporciona la ventaja de ofrecer un mínimo conjunto de servicios de forma gratuita, que podría servir para realizar una primera aproximación al problema y controlar su funcionamiento. Aún así, se debe tener en cuenta el crecimiento y evolución futuros, ya que si la plataforma fuese incorporando un gran número de usuarios y aplicaciones, se debe garantizar el mejor rendimiento posible y no sería suficiente el conjunto de propiedades gratuitas proporcionado primeramente.

En cuanto a herramientas y funcionamiento, Windows Azure tiene la contraprestación de que muchos de sus desarrollos se encuentran optimizados para herramientas de Microsoft, como Visual Studio, hecho que puede llegar a limitar algunas opciones. Por el contrario tanto la oferta de Google como la de Amazon ofrecen mucha libertad en este sentido.

Otro aspecto importante es el nivel de desarrollo de las plataformas. En este sentido, Google es la empresa que lleva menor tiempo en el mercado ofreciendo este tipo de infraestructura y aún tiene componentes y servicios en fases de prueba o proyectados como futuros desarrollos. Este hecho hace que retroceda un paso con respecto a las otras dos, a pesar de que los servicios y condiciones que ofrece son iguales o incluso superiores en algún caso. Por otro lado, Amazon fue pionera en este campo, su rendimiento es excelente y durante los últimos meses han ido aumentando su capacidad y mejorando el rendimiento de los productos ofrecidos. Su nivel de madurez es mayor y por tanto, su nivel de eficacia también lo es.

En cuanto a facilidad de uso, las características de Google App Engine superan a las de sus competidores. Su configuración es más sencilla y requiere de un menor número de acciones para conseguir un sistema funcional. Esto tiene la contrapartida de que la sencillez de configuración por parte de Google reside en gran parte en que se limitan considerablemente los parámetros configurables, ofreciendo un menor nivel de personalización. En ese sentido, Amazon es más complejo, pero con un conocimiento adecuado se pueden establecer configuraciones mucho más ajustadas y adaptadas a las necesidades de cada usuario.

Como se puede evidenciar, todas estas infraestructuras se encuentran muy parejas en cuanto a propiedades y precios, lo que dificulta el proceso de toma de decisión en el que se inclinará la balanza en función de pequeños detalles.

Finalmente se ha decidido que la infraestructura más adecuada para el sistema propuesto es la proporcionada por Amazon EC2. La elección se ha basado en una mayor madurez y experiencia en el sector así como en la fiabilidad que ofrece y los servicios de mantenimiento, almacenamiento con S3 y un mayor número de opciones complementarias para la gestión, monitorización y configuración de la plataforma. Google App Engine se plantea como una gran opción para el futuro, cuando además de alojar aplicaciones web, terminen de diseñar componentes adicionales y se permitan sistemas de gestión más complejos y completos que añadan más valor añadido a la estructura, ya que sus propiedades de sencillez de uso y la posibilidad e iniciar las aplicaciones de forma gratuita resulta muy interesante. Por último, Windows Azure se queda algo atrás debido a algunas de sus dependencias. Bien es cierto que permite también sistemas Linux y desarrollos con otras herramientas, sin embargo el hecho de encontrarse optimizado sólo para algunas de ellas, hace que estas limitaciones sean determinantes en una comparación tan igualada como la que se ha presentado.

Sin duda, el sistema podría desarrollarse en cualquiera de las infraestructuras expuestas debido a sus potenciales, sin embargo, debido a las características presentes y futuras que se desean alcanzar con el mismo y a las prioridades establecidas para su desarrollo, la balanza se ha inclinado a favor de Amazon EC2 por ser empresa pionera en el sector, por su madurez y por superar, en media, las propiedades ofrecidas por el resto de sus competidores.

4.7 Funcionalidad del sistema

Una vez conocidos los diferentes elementos que forman parte del sistema completo donde se produce la acción del lenguaje objeto de esta investigación, esta sección presenta un ejemplo detallado de la funcionalidad del sistema y de la forma en la que el lenguaje se integra con la plataforma de computación en nube y las aplicaciones existentes en ella. De esta manera es posible entender de manera completa, clara y práctica el alcance de la investigación realizada.

La siguiente secuencia de pasos muestra el ciclo de vida de una aplicación desde que es subida a la nube hasta que intercambia datos con otra aplicación dentro de esa misma nube:

1. **Registro individuo/organización en la plataforma de Cloud Computing**. El primer paso se produce cuando una persona/organización desea subir su aplicación a la plataforma de Cloud Computing. Lo primero que se debe hacer en este caso es registrarse en la plataforma y proporcionar información personal y económica que permitan identificar de forma unívoca al propietario y beneficiarse así de todas sus propiedades. Estos datos quedarán almacenados en la base de datos comentada anteriormente, en la capa de persistencia.

2. **Registrar aplicación**. Una vez dado de alta el propietario o tenant, se proporcionan las opciones necesarias para la subida y etiquetación de aplicaciones en la plataforma a través del "Complex Application Editor and Mapper". El proceso se inicia accediendo a un formulario en el que se solicitará introducir información relativa a la propia aplicación (campos, etiquetas, descripción, etc.), además de, por supuesto, la propia aplicación y las bases de datos o elementos adicionales que precise para funcionar correctamente. Adicionalmente a la opción de rellenar el formulario, se puede utilizar un fichero con extensión ".carl" (como el que se comentaba en la sección 3.3.3) en el que se indique toda la información necesaria. Esta última opción es muy recomendable para aplicaciones con un gran número de campos, ya que simplifica el proceso de carga no teniendo que añadir uno a uno los campos de los que precise, facilitando también las modificaciones posteriores.

   La información a introducir constará de indicar el tipo de aplicaciones a las que pertenece, los campos presentes en la aplicación, sus tipos y su relación con la ontología del dominio, es decir, especificar con qué campo del vocabulario GoodRelations se corresponde cada uno de ellos. Es importante resaltar que el propietario no debe modificar en ningún momento su aplicación, simplemente indicar en el formulario el campo con el que se corresponde en la ontología del dominio.

3. **Almacenamiento en la ontología**. Una vez introducida y aceptada la información, CARL la recoge en la ontología que almacena el conocimiento adquirido mediante las etiquetas proporcionadas por el propietario de la aplicación. Desde este momento la aplicación será totalmente funcional en la plataforma y cualquier usuario podrá suscribirse a ella.

4. **Establecimiento de conexión**. Una vez operativa, la aplicación subida requiere de cierta información de la que carece. Para conseguirla debe establecer comunicación con el sistema. La aplicación origen contacta con la "Interoperability Interface" a través de servicios web donde envía la información que desea y los datos necesarios para su localización. En este momento se inicia el proceso de interoperabilidad capturando esta información y enviándola al motor para que indique el siguiente paso a realizar.
5. **Actuación del motor e interacción con el razonador**. Cuando el motor recibe la información de la aplicación origen, se suceden una serie de pasos que se comentan a continuación:
    a. La información pasa en primer lugar por el lexer que pone el mensaje en memoria y lo divide en tokens.
    b. La salida del lexer va al parser que selecciona aquellos tokens relevantes para el proceso (mediante el estudio de los inputs y outputs del WSDL que describe el mensaje).
    c. Una vez se han seleccionado los datos necesarios para el intercambio, se envían al analizador semántico quien envía la información recibida al intérprete para que traduzca el contenido y sea comprensible por el razonador.
    d. El intérprete establece el primer contacto con el razonador para que éste le proporcione la relación existente entre la información enviada por la aplicación y la ontología del dominio mediante consultas SPARQL a la ontología.
    e. Cuando el analizador semántico recibe esta información se encarga de formar un mensaje con sintaxis CARL con las equivalencias entre los campos involucrados.
    f. En este momento, la información llega de nuevo al intérprete que se encarga de traducir las sentencias CARL a OWL-DL de forma que sean entendibles por el razonador.
    g. Realizada esta traducción, el razonador se encarga de obtener los datos requeridos a través de consultas SPARQL a la ontología. En caso de que no se obtengan resultados, se aplicarán las funciones de transformación a los datos de localización hasta encontrar un resultado o agotar las posibilidades.
    h. Al conseguir esta información se devuelve al motor que estudia el tipo de dato del campo y, si es preciso, decide y ejecuta la función de transformación adecuada.
6. **Interoperabilidad alcanzada**. Con los campos ya transformados al formato adecuado, estos son comunicados a la "Interoperability Interface" para que, vía servicios web, sean enviados a la aplicación de origen y pueda disponer de ellos para la tarea deseada.

Como se puede comprobar, el proceso completo desde que se registra una aplicación hasta que se alcanza la interoperabilidad consta de un pequeño número de pasos, la mayoría de los cuales resultan invisibles para el usuario. El sistema propuesto presenta una solución adecuada al problema y permite simplificar la secuencia de acciones a realizar gracias a la actuación del lenguaje, la gestión del motor y la interacción entre el razonador y la ontología.

A continuación la Figura 54 muestra un diagrama de flujo que representa gráficamente la secuencia de pasos explicada con anterioridad:

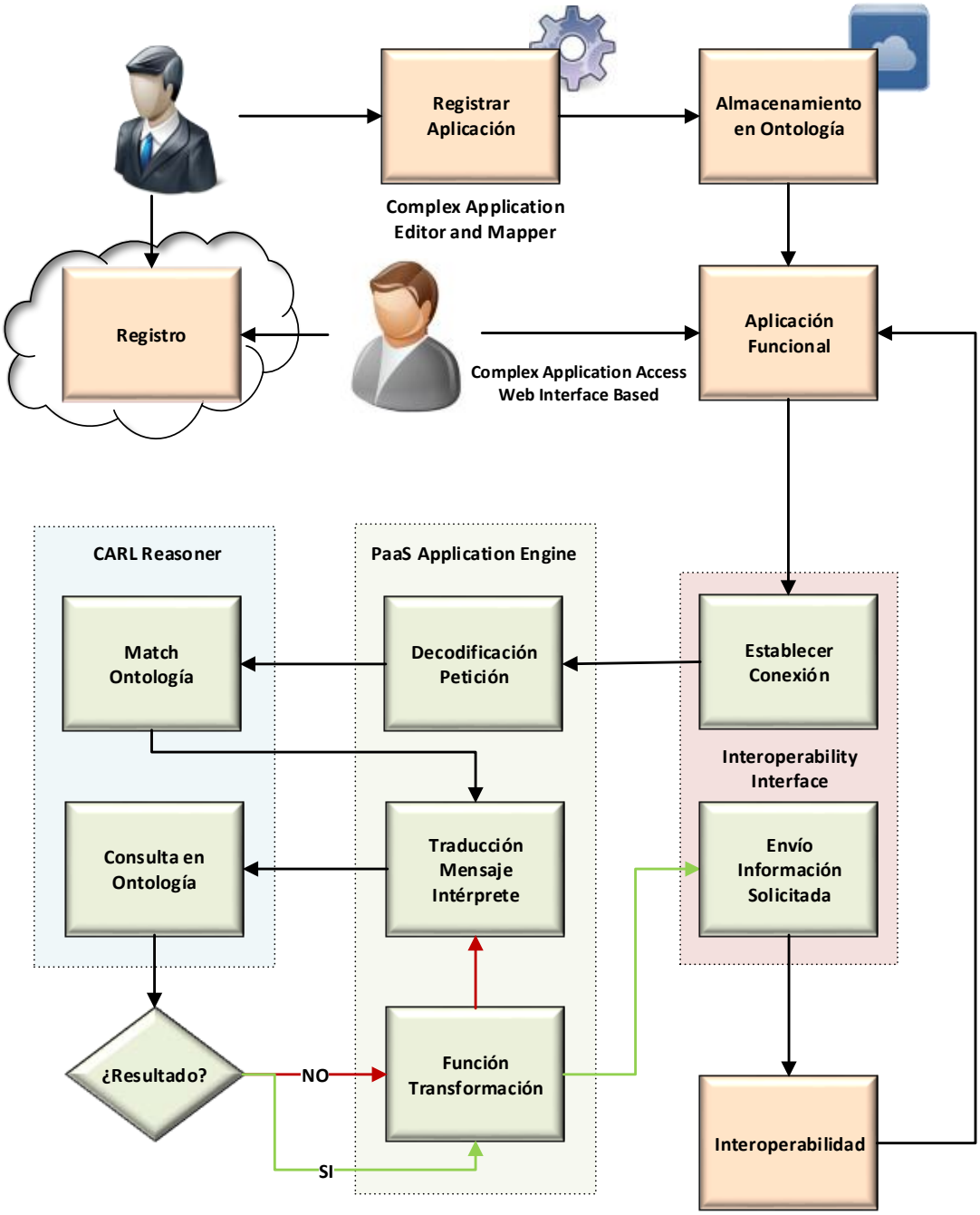


Figura 54. Diagrama de flujo del funcionamiento general del sistema

Como se puede observar, en la figura se muestran las interacciones más relevantes que se suceden en el sistema así como el componente responsable de su realización.

Aunque no formen parte de la estructura real del sistema, tanto los propietarios de las aplicaciones (tenants) como los usuarios finales se reflejan en el diagrama para indicar el momento en el que interactúan con el sistema y el componente con el que lo hacen. Por supuesto, no podrán participar en el sistema si no se ha realizado el registro previo en el mismo.

Para diferenciar los tipos de tareas que se realizan, se muestran en la parte superior aspectos de registro e introducción de información, previos a los procesos de consecución de la interoperabilidad (acciones ilustradas en color naranja), y por otro lado la secuencia de procesos internos que realiza el sistema de manera automática para alcanzar la interoperabilidad (actividades ilustradas en color verde). Para estos últimos se muestran además sombreados en diferentes colores las actividades que realiza cada componente del sistema, estableciendo una diferenciación modular para indicar las responsabilidades de cada elemento del sistema.

Con esta representación se consigue un buen entendimiento del funcionamiento del sistema que clarifica las diferentes tareas que se llevan a cabo y sobre qué componente recae la responsabilidad de la misma, ofreciendo una visión clara de las diferentes fases por las que pasa la información dentro del sistema.

5 Evaluación y validación

Este capítulo describe el proceso de validación que se ha llevado a cabo a partir de las hipótesis planteadas. La validación resulta una fase de gran importancia para los investigadores ya que permite estudiar el alcance del trabajo y demostrar si las hipótesis de las que parte son probadas como verdaderas, falsas o por el contrario no se pueden demostrar.

Tras haberse expuesto la totalidad de los objetivos, las decisiones de diseño tomadas para alcanzarlos y los diferentes conceptos utilizados, este capítulo se encarga de evaluar los resultados alcanzados por el sistema propuesto. Para ello se establece una metodología de validación adaptada a cada hipótesis que permita una demostración más precisa y fiable.

Para finalizar el capítulo se presenta un sumario o discusión final que expone unas conclusiones basadas en los resultados obtenidos y confirman la viabilidad de la solución propuesta.

5.1 Introducción

En el capítulo 4 se han definido y justificado las diferentes decisiones de diseño que permiten afrontar los retos planteados por la investigación. Esta justificación permite asegurar de forma teórica la adecuación de los diseños propuestos en relación a la solución planteada, cuyo objetivo final es la validación de las hipótesis que han guiado la totalidad del proceso de investigación.

Las hipótesis expuestas en la sección 3.1 plantean la creación de un lenguaje semántico para entornos de Cloud Computing capaz de capturar el conocimiento de las aplicaciones alojadas en estas estructuras permitiendo su interoperabilidad de manera automática. Estas hipótesis se han dividido en principales y secundarias de manera que se puedan establecer una serie de objetivos parciales que guíen el proceso y permitan alcanzar adecuadamente las metas de la tesis doctoral.

A lo largo de las siguientes secciones se detallan los procesos seguidos para demostrar cada una de las hipótesis de forma que se proporcione una información objetiva sobre los resultados alcanzados por la investigación, probando así el éxito de la misma. Además para poder validar determinados aspectos del lenguaje, se ha desarrollado un sistema que permite mostrar el correcto funcionamiento del mismo.

5.2 Metodología de validación

La metodología de validación establece los criterios que se seguirán a la hora de evaluar los diferentes aspectos relevantes que permitan demostrar la validez y eficiencia del sistema derivado de la investigación realizada. Para ello y como se comentaba con anterioridad, se tomarán como base las hipótesis planteadas en la sección 3.1.

Debido al carácter multidisciplinar del trabajo y a los conceptos heterogéneos representados en las hipótesis, no se ha podido establecer una metodología general de demostración que permita cubrir de forma objetiva todas las hipótesis y seguir así un mismo patrón de validación. Por esta razón se ha decidido estudiar cada hipótesis de manera aislada e individual y elegir para cada una de ellas el método de demostración más adecuado y adaptado a sus características particulares, de esta manera se podrá proporcionar un mejor estudio de los resultados arrojados por el sistema en cada caso.

A pesar de que no es posible establecer una metodología de validación común para todas las hipótesis, sí que se han seguido una serie de pasos para la demostración de todas ellas que permiten determinar el mejor proceso de validación de cada hipótesis y presentar una serie de resultados que garanticen su correcto enfoque y funcionamiento. La siguiente figura muestra la secuencia de pasos realizados para conseguir una evaluación objetiva:

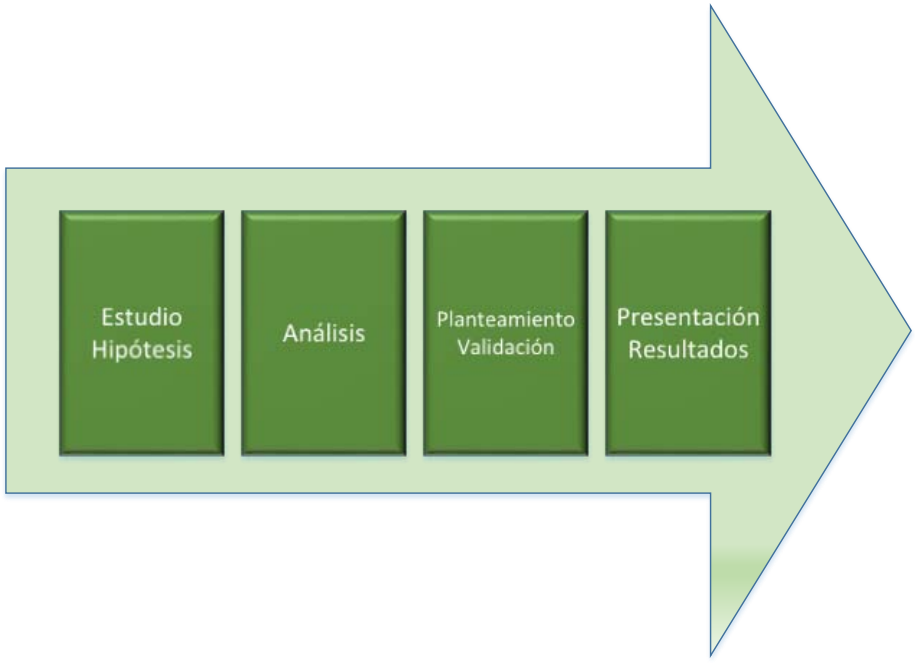


Figura 55. Fases para la demostración de hipótesis

Como se puede comprobar en la Figura 55 el proceso a seguir para la demostración de cada hipótesis de forma individual se divide en cuatro fases:

1. Estudio de la hipótesis: El estudio preliminar de la hipótesis permitirá conocer la naturaleza de la misma y de esta manera determinar si el enfoque de la validación debe tener carácter teórico o práctico y plantear una serie de alternativas (en caso de que las haya) que permitan una demostración más fiable y adecuada a la hipótesis particular.
2. Análisis: Una vez conocidas las posibles alternativas, analizar los pros y contras de cada una de ellas y realizar un proceso de toma de decisión que dé como resultado la elección de un método de validación eficaz y objetivo.
3. Planteamiento de la validación: Tras seleccionar el proceso de demostración que mejor se adapte a las características de la hipótesis en particular, se presenta el modelo que permita al lector entender de forma clara las implicaciones del trabajo realizado así como observar que la hipótesis ha sido demostrada en su totalidad.
4. Presentación de resultados: Como última fase se hace necesario presentar los resultados obtenidos y una explicación razonada de la forma en la que se ha llegado a ellos y lo que supone el avance conseguido (si lo hubiese) para el campo en el que se encuentre.

Siguiendo los pasos previamente mencionados y cubriendo los contenidos de cada una de las fases, el proceso de validación completo permitirá extraer unas conclusiones serias, objetivas y fundamentadas que proporcionarán toda la información necesaria para comprender el alcance de la investigación realizada y las aportaciones de la misma.

A continuación se presenta el proceso de validación de cada una de las hipótesis que han guiado el desarrollo de esta tesis doctoral.

5.3 Validación de hipótesis

A lo largo de esta sección se presenta el estudio y análisis particular de cada una de las hipótesis siguiendo las fases detalladas en la sección anterior. Con ello se alcanza el aseguramiento de unos resultados fiables y se permite demostrar las hipótesis expuestas, objetivo principal de la investigación desarrollada.

Seguidamente se presenta cada una de las hipótesis del trabajo y su validación correspondiente a través del sistema global desarrollado para ello. Esta validación se dividirá en los bloques presentados en la sección 3.1. Partiendo de esta organización, se planteará la hipótesis principal de cada bloque y se validará cada una de las hipótesis secundarias en las que se descompone quedando, por consiguiente, validada la hipótesis principal al estar formada por la unión de dichos objetivos parciales.

5.3.1 **Hipótesis 1**

El primer bloque viene definido por la hipótesis principal que enuncia que *un lenguaje declarativo semántico basado en ontologías soluciona el problema de la interoperabilidad entre aplicaciones dentro de un entorno de Cloud Computing*. De esta manera, esta hipótesis se centra en el problema de la creación del lenguaje y la aportación de las tecnologías semánticas para solucionar el problema de la interoperabilidad.

Para la validación de esta hipótesis se debe demostrar la viabilidad de creación de un lenguaje de estas características y que el empleo de tecnologías propias de la Web Semántica junto con el lenguaje pueden resolver el problema planteado. Por esta razón se han creado las siguientes hipótesis secundarias que dividen el problema y permiten estudiar cada aspecto de forma aislada y detallada para proporcionar un proceso de validación adecuado en cada caso y cuya demostración conjunta permite validar la hipótesis principal planteada en este bloque.

5.3.1.1 Hipótesis 1.1

El enunciado de esta hipótesis es que *es posible desarrollar un lenguaje basado en ontologías*. Para probar esta afirmación es necesario señalar la presencia de otros lenguajes que sigan estas mismas directrices y que, como CARL, basen su funcionamiento en ontologías.

El primer ejemplo de estos lenguajes lo encontramos con WSML (previamente mencionado en la sección 2.6.2) que proporciona la semántica y la sintaxis en WSMO (De et al., 2005), es decir, permite describir formalmente sus elementos como ontologías, servicios web semánticos, mediadores, etc. Posee diferentes variantes en función de distintos niveles de expresividad necesarios para cada problema particular y nace con la intención de solventar los problemas presentes en OWL (Bruijn et al., 2005) y emplearse como lenguaje unificado para WSMO.

Otros ejemplos de lenguajes que se utilizan para la creación de ontologías los encontramos en OWL (ampliamente explicado en la sección 2.4.3.2) y en su antecesor DAML+OIL (Horrocks, 2002), mencionado en la misma sección. Un ejemplo adicional aparece en CoOl (Strang, Linnhoff-Popien, & Frank, 2003), un lenguaje basado en ontologías que además se encuentra dirigido hacia la interoperabilidad contextual. Todos estos lenguajes basan su funcionamiento en torno a las ontologías, al igual que CARL, y demuestran la existencia de este tipo de lenguajes.

La diferencia de CARL con estos lenguajes es que CARL utiliza la semántica de OWL (aunque no la sintaxis) para describir los elementos presentes en una estructura de Cloud Computing, que actúa como "jardín vallado" limitando el entorno. Precisamente por este principio, eran objetivos principales que el diseño de CARL fuera sencillo y específico, permitiendo la máxima semántica con la mínima definición del lenguaje, diferenciándose así de los lenguajes anteriores. Por ello, sería posible crear otros lenguajes similares a CARL mediante WSML, OWL o DAML+OIL para describir determinados dominios.

Además de los ejemplos presentados, se ha podido demostrar el correcto funcionamiento de CARL para la descripción de ontologías y la comunicación automática entre aplicaciones heterogéneas, objetivo principal de la investigación y hecho que permite demostrar la hipótesis objeto de esta sección. El sistema desarrollado para la validación permite mostrar distintas vistas del correcto funcionamiento del lenguaje y asegurar así su potencial y usabilidad. A continuación se muestran algunos ejemplos que permiten comprobarlo a través de vistas web del sistema:

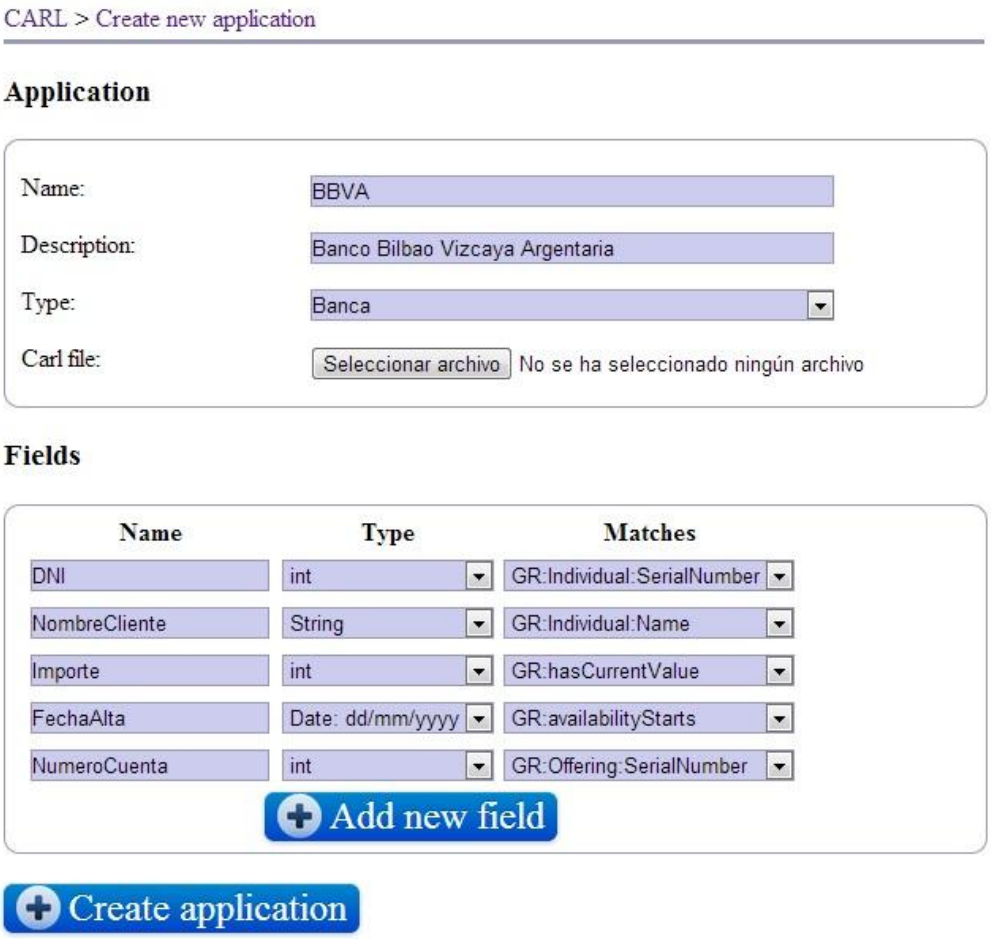


Figura 56. Insertar nueva aplicación

En la Figura 56 se observan las formas en las que se pueden crear aplicaciones e insertarlas en el sistema. De esta manera, cuando se introducen los campos y sus relaciones con la ontología del dominio (ya sea a través del formulario o adjuntando un archivo ".carl"), estos se introducen en la ontología, para así proporcionar la información necesaria para el funcionamiento del lenguaje y sin la cual el lenguaje semántico no tendría sentido.

A continuación se muestran los datos de la misma aplicación extraídos de la ontología (ver Figura 57). Una vez que se ha finalizado el proceso de creación de la aplicación e inserción de datos, estos pueden ser consultados y editados en cualquier momento.

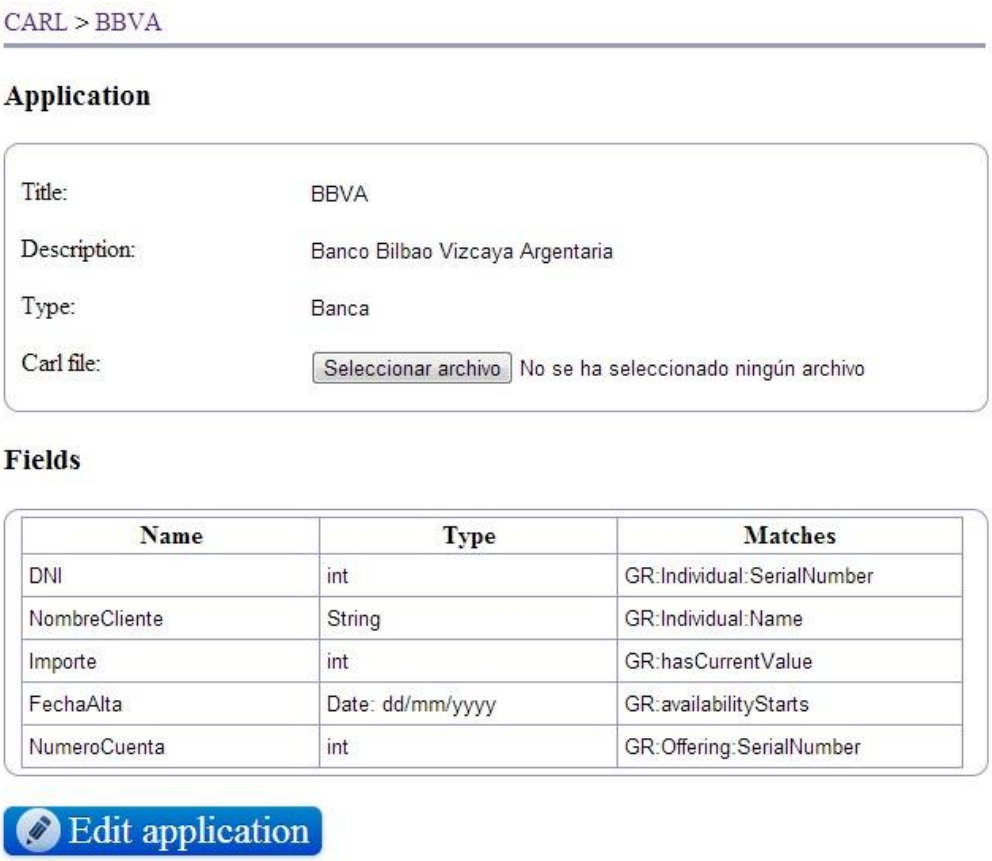


Figura 57. Editar datos aplicación

Como se puede comprobar, al consultar los datos en la ontología, estos se obtienen correctamente y muestran la información que ha sido previamente introducida por el propietario de la aplicación. Esta captura del sistema de validación pretende simular el comportamiento del lenguaje a la hora de acceder a los datos contenidos en la ontología para poder hacer uso de ellos.

Con la demostración teórica y práctica presentada, se prueba que la hipótesis se cumple, ya que es posible el diseño de un lenguaje que base su funcionamiento en ontologías y que utilice ese conocimiento para aportar valor añadido a las operaciones que realice. A lo largo del documento y de esta sección se han expuesto ejemplos de otros lenguajes que se fundamentan en los mismos principios, sus limitaciones y las oportunidades que presenta CARL frente a ellos. Además, la demostración práctica de que mediante el lenguaje se pueden introducir y consultar los datos en la ontología, complementan las explicaciones teóricas y muestran su correcto funcionamiento, completando así la validación de esta hipótesis.

5.3.1.2 Hipótesis 1.2

La hipótesis en cuestión enuncia que *un lenguaje basado en tecnologías semánticas permite solucionar el problema de la heterogeneidad e incompatibilidad entre aplicaciones.* Como se ha explicado previamente en la sección 3.3, en esta investigación el problema se soluciona mediante el conocimiento de las aplicaciones capturado por el lenguaje y almacenado en la ontología y las funciones de conversión que el sistema se encarga de gestionar.

Existen distintos tipos de heterogeneidades como pueden ser de procesos (Puckett & Hensher, 2009), protocolos (Bottaro & Gérodolle, 2008), datos (Kim & Seo, 1991), etc. Como se ha detallado a lo largo de todo el documento, este trabajo se encarga de solventar los problemas que surgen a nivel de datos, ya que para el sistema diseñado los procesos y los protocolos no suponen un problema ni constituyen un elemento central. Las incompatibilidades de los datos surgen cuando varias aplicaciones desean intercambiar información para de esta manera poder acceder a datos de los que carecen sin necesidad de interacción alguna por parte del usuario. Cada aplicación posee un conjunto de campos con sus formatos correspondientes. Estos campos son libres con el fin de que el propietario de la aplicación deba hacer el menor número de modificaciones posibles sobre ella para actuar en la plataforma de computación en nube. Aún con esto, sí debe establecer una relación de concordancia de sus campos con los presentes en la ontología del dominio (GoodRelations) al introducir su aplicación en la plataforma.

Una vez que el propietario ha realizado este paso, la ontología contiene información suficiente de las aplicaciones como para que el lenguaje haga uso de ella para automatizar el proceso de intercambio de información, momento en el que pueden surgir las incompatibilidades. La libertad a la hora de asignar los formatos genera que un mismo campo de dos aplicaciones pueda tener distintas representaciones tanto a nivel de nombre como de tipo de datos asignado. Es aquí cuando el sistema, a través de la información capturada por el lenguaje, se percata de esta incompatibilidad y la solventa.

Para demostrar que el lenguaje es capaz de solucionar estos problemas, se ha realizado un batería de pruebas de 20 aplicaciones con 15 campos cada una y se ha realizado el proceso de intercambio de información entre ellas. A continuación se muestran dos capturas extraídas de esta batería de pruebas que demuestran que la conversión de datos se produce correctamente y que permiten a la aplicación destino hacer uso de dicha información. Las capturas seleccionadas muestran el proceso de intercambio de información entre las aplicaciones de “Iberia” y “BBVA” y se han obtenido gracias al sistema desarrollado para la validación y que permite visualizar el proceso interno de intercambio de datos:

CARL > Data exchanged between Iberia and BBVA

Iberia					BBVA			
Field	**Matches**	**Type**	**Value**		**Field**	**Matches**	**Type**	**Value**
Name	GR:Individual:Name	String	Miguel	→	NombreCliente	GR:Individual:Name	String	Miguel
Identifier	GR:Individual:SerialNumber	String	47451012L	→	DNI	GR:Individual:SerialNumber	int	47451012

Figura 58. Resolución de heterogeneidad entre Iberia y BBVA

En el ejemplo se muestra la situación en la que la aplicación de "Iberia" desea conocer cierta información de un cliente contenida específicamente en la aplicación del "BBVA". Es entonces cuando se pasa la información de localización al sistema que debe localizar al cliente objeto de la consulta de la aplicación origen (en este caso, Iberia) en la información de la aplicación destino (BBVA) para, posteriormente, realizar conversiones de datos, si corresponde, haciendo que los formatos sean compatibles. Es este punto el que se muestra en la Figura 58, justo el momento en el que la información sale del analizador semántico, donde se han establecido las correspondencias entre los campos y GoodRelations, pero aún no se ha realizado la consulta para conseguir el dato deseado. La aplicación de Iberia pasa sus datos "Name" e "Identifier" para localizar al cliente en la aplicación del BBVA. En este caso, como el tipo del identificador en Iberia no coincide con el correspondiente identificador en BBVA, la consulta SPARQL que realizará el razonador no devolverá ningún resultado para el campo buscado. Por este motivo, el motor realiza las transformaciones correspondientes para adaptar el tipo del identificador al formato necesario. Una vez aplicada la función de transformación, la información pasa al intérprete quien traduce las sentencias CARL a OWL para que el razonador esté en condiciones de obtener el dato en cuestión. En ese momento ya se obtendrán resultados y se podrá continuar hasta el siguiente paso, como se procede habitualmente.

Esta situación se refleja en la

Figura 59 donde se puede observar como "BBVA" envía el valor del número de cuenta del cliente deseado a Iberia. Como existe incompatibilidad en el tipo de datos de los campos, CARL se encarga de solucionarlo y aplicar la función de transformación correspondiente (para la visualización se ha añadido el carácter "#" para probar que la conversión se ha llevado a cabo). De esta manera, se completa el intercambio de información solventando los problemas de heterogeneidad en los datos comentados previamente.

CARL > Data exchanged between BBVA and Iberia

BBVA					Iberia			
Field	**Matches**	**Type**	**Value**		**Field**	**Matches**	**Type**	**Value**
NumeroCuenta	GR:Offering:SerialNumber	int	6372372736278273	→	CuentaCliente	GR:Offering:SerialNumber	String	6372372736278273#

Figura 59. Intercambio de información heterogénea entre BBVA e Iberia

Es necesario hacer notar que para que se produzca el intercambio de información, las aplicaciones deben tener un nexo en común que lo permita, es decir, que ambas posean un campo relacionado con el mismo campo existente en GoodRelations. No se pueden realizar transformaciones entre campos de diferente naturaleza. Por ello, en los procesos de intercambio de datos se pueden dar varias situaciones:

1. *El campo/campos que solicita una aplicación no encuentran sus homólogos en la aplicación destino*. Si en un proceso de intercambio las aplicaciones involucradas no poseen campos relacionados con el mismo presente en la ontología del dominio, éste no puede efectuarse y se dará por finalizado informando a la aplicación que origina el proceso de intercambio. La Figura 60 muestra en forma de captura la respuesta del sistema cuando se intenta intercambiar información no transformable entre dos aplicaciones:

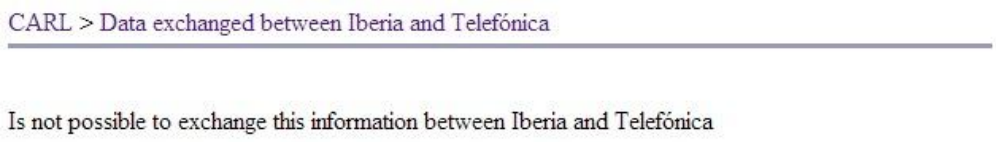


Figura 60. Mensaje de imposibilidad de transformación de datos entre aplicaciones

   En esta situación en particular puede darse el escenario comentado con anterioridad, en la que a la hora de realizar la consulta, la información que sirve para localizar el dato deseado sea de tipo diferente al de las aplicaciones donde se pueda localizar. En este caso el sistema procede aplicando una tras otra todas las funciones de conversión compatibles que toman como origen el tipo de dato utilizado para la consulta. Este proceso continúa hasta que se obtiene un resultado, momento a partir del cual se procede normalmente. En caso de agotar todas las posibles funciones de transformación y no conseguir un resultado, se mostrará un mensaje similar al de la Figura 60.

2. *El campo/campos que solicita una aplicación posee homólogos en la aplicación destino y necesitan ser transformados*. Es la situación principal para el funcionamiento del sistema. Al originarse el proceso, el sistema se encarga de buscar el formato de ambas aplicaciones y llamar al motor para ejecutar la función de conversión adecuada, permitiendo así que el proceso se complete exitosamente. Un ejemplo claro de esta situación se encuentra en la Figura 59 (mostrada previamente) donde se realiza una transformación en el formato del número de cuenta entre dos aplicaciones.

3. *El campo/campos que solicita una aplicación posee homólogos en la aplicación destino pero no necesitan transformación*. Puede darse el caso de que al iniciarse el proceso de intercambio de información el campo/campos a intercambiar posean el mismo formato en ambas aplicaciones. De esta manera el sistema indicará que no es

necesaria la conversión de datos y el proceso se convertirá en un mero intercambio de información. El campo “Name” de la Figura 58 ilustra esta posibilidad, ya que en ambos casos se encuentra almacenado como una cadena de caracteres y no necesita de conversión alguna si se diese el caso de su intercambio.

Una vez mostrados los experimentos realizados, se puede comprobar que el lenguaje reacciona adecuadamente a las incompatibilidades a nivel de datos y permite solucionarlas de forma adecuada. Por ello, se puede concluir que la hipótesis ha quedado demostrada y un lenguaje basado en tecnologías semánticas se presenta como una solución válida para solucionar el problema de la heterogeneidad a nivel de datos.

5.3.2 **Hipótesis 2**

La hipótesis principal que define el segundo bloque de hipótesis afirma que *dotar de semántica a un lenguaje confiere propiedades ventajosas para la interrelación entre aplicaciones y la localización de información*. Con esta hipótesis se desea demostrar el potencial y las posibilidades que añaden las tecnologías semánticas al rendimiento del lenguaje. Con ello se proporciona un valor añadido a la solución ofrecida en cuanto a la interoperabilidad, ya que no sólo permite afrontar el problema sin más, si no que lo hace de una forma eficiente y aportando nuevas herramientas y opciones para aumentar las posibilidades del sistema.

Por estos motivos, la validación de esta hipótesis debe centrarse en demostrar las ventajas que conlleva la utilización de la semántica en el problema objeto de estudio a distintos niveles. Estas ventajas que se obtienen se reflejan en las hipótesis secundarias que guiarán esta parte de la validación y que se descomponen en sus propiedades ventajosas relacionadas con la propia gestión y acceso a la información, con su posibilidad de automatizar los procesos de comunicación necesarios y con la optimización del número de transacciones o transformaciones a realizar durante los procesos de intercambio de información.

A continuación se presenta la validación de las hipótesis secundarias correspondientes a este bloque.

5.3.2.1 Hipótesis 2.1

La hipótesis a demostrar en este apartado es que u*tilizar un medio de representación del conocimiento semántico proporciona ventajas a la hora de acceder, modificar y consultar la información relativa al intercambio de datos entre dos o varias aplicaciones en entornos de computación en nube*. Para la validación de esta hipótesis se realizará un enfoque teórico que probará que las propiedades de un lenguaje semántico benefician su funcionamiento y rendimiento.

En primer lugar, esta afirmación es demostrable a través de tres principios inherentes a la definición y esencia de un lenguaje:

1. El primer principio es el de la lógica y exclusividad dentro del lenguaje. Este principio enuncia que todo lenguaje debe estar respaldado por una base formal (Hoare, 1978), es decir, por un conjunto de propiedades de naturaleza matemática que permita eliminar la ambigüedad y permita (o no) propiedades como la transitiva, asociativa o distributiva en los diferentes términos del lenguaje.
2. El segundo principio corresponde a la propia validación del lenguaje (Chomsky, 1965). Este principio imprime carácter unívoco a determinadas expresiones y además asegura que la comunidad de usuarios que utiliza el lenguaje comparte unos principios, identidades culturales y semblanzas que lo validan como medio de comunicación.
3. El tercer principio es el de naturaleza única del lenguaje. Precisamente en el contexto de lenguajes para un dominio relacionado con esta tesis doctoral, el de los lenguajes que describen Servicios Web Semánticos (Fensel et al., 2006), se realizaron numerosos trabajos de comparación y vertebración entre varios lenguajes que tratan de describir y expresar los mismos elementos del dominio pero con diferentes niveles de expresividad. Por ejemplo, en (Lausen, de Bruijn, Polleres, & Fensel, 2005) se muestra un trabajo sobre un lenguaje con la misma sintaxis pero tres semánticas diferentes basadas en Datalog, Description Logic y Frame Logic. Otro ejemplo se encuentra en (Lara, Roman, Polleres, & Fensel, 2004) donde se comparan dos lenguajes con el mismo propósito que también difieren en la semántica, siendo WSMO, un conjunto de lenguajes con la misma sintaxis y OWL-S uno solo con una sintaxis y semántica predeterminada. La clave es que la semántica de cada lenguaje lo complementa y le añade propiedades formales, manteniendo su naturaleza única.

Estos principios demuestran que la semántica aplicada a los lenguajes les proporciona cualidades ventajosas. Si bien, desde otras perspectivas también es posible probar el aumento cualitativo que supone la agregación de la semántica.

SPARQL (Prud'hommeaux & Seaborne, 2008) es un lenguaje de consultas para RDF considerado como una tecnología clave para la Web Semántica (es una recomendación del W3C). Se puede utilizar para expresar consultas que permiten interrogar diversas fuentes de datos, si se almacenan de forma nativa como RDF o son definidos mediante vistas RDF a través de algún sistema middleware. Los resultados de sus consultas se pueden presentar como conjuntos de resultados o como grafos RDF. Además existen herramientas que pueden conectarse y construir de forma semiautomática consultas SPARQL. Por todo ello, la utilización de este lenguaje de consultas conlleva importantes ventajas que pueden ser aplicadas gracias al uso de las tecnologías semánticas:

- SPARQL es a la Web Semántica (o la Web en general) lo que SQL es a las bases de datos relacionales (en palabras del propio Tim Berners-Lee).

- Si se considera la Web Semántica una colección global de bases de datos, SPARQL puede hacer que esa colección sea concebida como una gran base de datos y permita obtener los beneficios propios de la centralización:

    - Centralización de información de múltiples sitios web (mashups).
    - Centralización de información de múltiples bases de datos corporativas.
    - Centralización de información entre sistemas internos y externos (outsourcing, bases de datos, web públicas, etc.).

- Existen numerosas tecnologías de bases de datos y resulta imposible seleccionar una de ellas a escala de la Web. RDF actúa como una lengua estándar en la que diferentes sistemas de bases de datos pueden ser representados y donde SPARQL actúa como lenguaje para esos datos. Como tal, SPARQL oculta los detalles de gestión de los servidores particulares y los detalles de estructuración. Esto reduce los costes y aumenta la robustez del software.

- SPARQL ahorra costes y tiempo de desarrollo permitiendo a las aplicaciones trabajar sólo con los datos en los que están interesados (en contraposición con la opción de descargar toda la información y emplear tiempo y dinero en desarrollo para extraer los bits relevantes de información). Para ilustrar esta situación se muestra el siguiente ejemplo:
    - Se desea conocer la población de las ciudades, su área y el coste de un billete de autobús para determinar si existe una relación entre la densidad de población y el coste del transporte público.
    - Sin la posibilidad de utilizar SPARQL, se tendría que realizar una primera consulta a las páginas de las ciudades en Wikipedia, una segunda para obtener el precio del billete de otra fuente y después codificar para extraer la población, área y tasas de autobús para cada ciudad.
    - Mediante SPARQL, esta tarea puede completarse escribiendo una única consulta SPARQL (sin necesidad de código adicional) que centraliza la apropiada fuente de datos.

- SPARQL puede trabajar con otros estándares incluyendo RDF, XML, HTTP y WSDL, lo que permite la reutilización software y herramientas existentes y facilita la interoperabilidad con otros sistemas.

Además de todo este potencial que aporta reduciendo los costes, aumentando la eficacia en las búsquedas y simplificando las consultas, SPARQL permite hacer consultas lógicas, que en dominios cerrados con el que se presenta en esta aplicación (jardín vallado establecido por la plataforma de Cloud Computing), funcionan mejor al permitir una localización más eficiente de los conceptos, al encontrarse estos almacenados en la ontología, a través de la cual puede navegar por los conceptos que forman el grafo y que se encuentran ligados por cualquier tipo de relación.

Otro aspecto relevante que permite demostrar la hipótesis planteada es que otras plataformas que ofrecen servicios de computación en nube como las conocidas Salesforce o Netsuite, no permiten realizar este tipo de operaciones y consultas lógicas al ser procedurales, no semánticos y no relativos. El caso de Netsuite se encuentra más enfocado a proporcionar soluciones a terceros (en todos los ámbitos de negocio) mediante su plataforma de Cloud Computing. Aun así, existen herramientas para el desarrollador que permiten cierta personalización y gestión de flujos de trabajo. En el caso de Salesforce, han creado un lenguaje de programación llamado Apex (Fisher, 2007) que permite a los desarrolladores ejecutar flujos y realizar un control de sentencias sobre la plataforma. Su sintaxis es similar a la de Java y permite añadir cierta lógica de negocio (aunque bastante limitada), crear servicios web y de correo, validación de objetos y creación de procesos de negocio complejos no soportados por los flujos de trabajo. Este lenguaje es compilado, almacenado y ejecutado completamente en la plataforma de Force.com. Como se puede deducir, ambos carecen de semántica y las operaciones lógicas que realizan no se encuentran basadas en inferencia sobre el conocimiento, si no que se limitan a personalizar copias de seguridad, añadir botones específicos, etc. Por todo ello CARL y el sistema completo presentado, suponen un gran avance en este tipo de plataformas, ya que la aportación de la semántica hace que se añadan estas interesantes propiedades mencionadas que resultan inalcanzables para el resto de aproximaciones.

Con toda la información expuesta se puede concluir que la hipótesis en cuestión ha quedado demostrada y que la semántica proporciona una serie de propiedades que confieren ventajas objetivas para el funcionamiento del lenguaje y la obtención de resultados. Desde la posibilidad de realizar inferencias lógicas, al aprovechamiento de las relaciones establecidas por los conceptos representados en la ontología y la posibilidad de emplear SPARQL como lenguaje de consultas (que además aumenta su eficacia en entornos cerrados) hacen que se permita un mejor acceso a la información, obtención más precisa de resultados y la extracción de conocimiento implícito. Además estas características diferencian este enfoque y resultados del resto de plataformas y trabajos en el área, añadiendo un componente de novedad y originalidad

5.3.2.2 Hipótesis 2.2

La siguiente hipótesis enuncia lo siguiente: *un lenguaje basado en tecnologías semánticas permite a las aplicaciones intercambiar información de manera automática, sin precisar interacción por parte del usuario*. Para la validación de esta hipótesis se adopta un enfoque teórico que finalizará con la demostración a través de un ejemplo práctico extraído del sistema de validación.

Para la demostración de esta hipótesis, se hace necesario definir qué se considera una interacción automática en el contexto de la investigación. Se entiende que un intercambio de información se realiza de manera automática cuando es posible que una aplicación obtenga un dato que necesita de otra aplicación sin necesidad de intervención humana. En otras palabras esta situación se traduce en que una aplicación debe saber qué dato necesita y proporcionar

información para conseguirla para que el sistema realice el conjunto de actividades necesario con el objetivo de proporcionarle la información deseada.

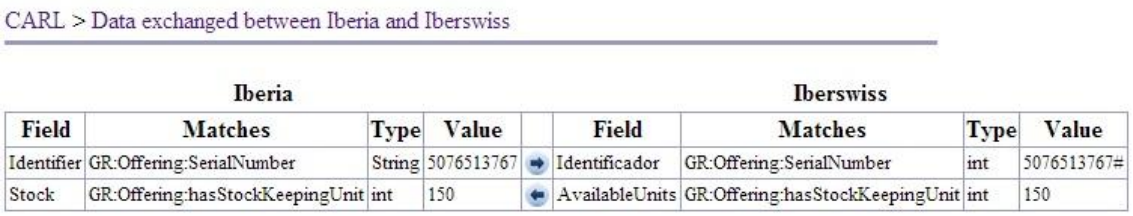
Gracias a la captura del conocimiento por parte de CARL, a la ontología y a la actuación del motor, es posible automatizar el proceso de intercambio de información. La automatización se realiza gracias al procesado de las peticiones y a los procesos realizados sobre el conocimiento almacenado de las aplicaciones. Estas tareas consisten en la gestión del proceso por parte del motor y en la realización de consultas SPARQL sobre la ontología llevadas a cabo por el razonador, procesos que permiten conocer los campos deseados. Una vez conocida la información que se desea localizar y se poseen los datos necesarios para localizarla, el sistema gestiona los procesos de comunicación y se producen las interacciones necesarias entre el razonador y la ontología para conseguir la información deseada. La automatización de este proceso resulta posible porque sólo a través de la definición de un lenguaje formal basado en lógica se pueden realizar procesos de inferencia y consultas lógicas como los mencionados. Esto deriva en la automatización de procesos y en la extracción de conocimiento, propiedades que añaden un relevante valor añadido a cualquier sistema. CARL, por tanto, posee estas propiedades y su semántica formal permite alcanzar el objetivo mencionado.

Bien es cierto que las tecnologías semánticas realizan habitualmente procesos semiautomáticos, es decir, supervisados por los usuarios que en muchos casos deben guiar el proceso o dar consentimiento a las operaciones. La definición propia de CARL y del sistema en el que se engloba hace que el consentimiento para realizar este tipo de operaciones se otorgue cuando el usuario se registra en la plataforma de computación en nube, aceptando los términos y condiciones, dando, por tanto, permiso a las aplicaciones y al sistema para utilizar sus datos cuando sea preciso. Con ello, CARL es libre para gestionar los procesos de comunicación desde que llega la petición de datos al sistema a través de los servicios web, hasta que se realiza se obtiene la información y su posterior transformación de datos si resulta necesario.

A continuación se comenta un ejemplo del intercambio automático que se produce entre las aplicaciones de “Iberia” e “Iberswiss”, ambas dadas de alta en el sistema:

1. El primer paso se produce desde la aplicación de Iberia, que a través de servicios web informa al sistema de que desea obtener el stock de un determinado producto que desean ofrecer en sus vuelos.
2. Cuando la información llega al sistema, el gestor de comunicaciones pasa la información al motor quien la traduce e interpreta.
3. Una vez interpretada esta información, el motor ordena al razonador que extraiga la información relevante a los conceptos deseados por Iberia.
4. Tras realizar las consultas apropiadas, el razonador devuelve al motor toda la información deseada relacionada con esos conceptos.
5. El motor comprueba si es necesario realizar alguna conversión. En caso afirmativo realiza la transformación en el formato de los datos extraídos por el razonador.
6. Una vez obtenidos los datos en el formato deseado, estos se vuelven a remitir al gestor de las comunicaciones que envía vía servicios web la información a la aplicación que la solicitó para que haga uso de ella.

Como se puede comprobar, esta secuencia de pasos refleja el proceso desde que Iberia solicita una determinada información hasta que esta se encuentra en la empresa Iberswiss, encargada de los catering de la empresa. En este caso concreto Iberia podría haber solicitado que el dato se extrajese de Iberswiss, limitando la búsqueda a los datos de esta aplicación. La ejecución de este ejemplo en el sistema desarrollado para la validación se puede observar en la siguiente figura:


CARL > Data exchanged between Iberia and Iberswiss

Iberia					Iberswiss			
Field	Matches	Type	Value		Field	Matches	Type	Value
Identifier	GR:Offering:SerialNumber	String	5076513767	→	Identificador	GR:Offering:SerialNumber	int	5076513767#
Stock	GR:Offering:hasStockKeepingUnit	int	150	←	AvailableUnits	GR:Offering:hasStockKeepingUnit	int	150

Figura 61. Proceso de intercambio automático de información entre Iberia e Iberswiss

En la Figura 61 se refleja el resultado de la sucesión de pasos comentada previamente. En ella se observa como a través del identificador del producto del que se desea saber el stock, se puede conocer esa información existente en la otra aplicación. El resultado final es que Iberswiss pone en conocimiento de Iberia de que dispone de 150 unidades de ese producto. A partir de ese momento Iberia puede actuar en consecuencia con la información en su poder.

Con la explicación teórica proporcionada y el ejemplo práctico mostrado a través del sistema de validación, se puede concluir que es posible el intercambio de información de manera automática entre las aplicaciones y que por tanto la hipótesis queda validada.

5.3.2.3 Hipótesis 2.3

Esta hipótesis expresa que l*a utilización de un estándar o medio de comunicación común permite reducir el número de transacciones durante los procesos de intercambio de información*. Por transformaciones se entiende el número de conversiones de datos que es necesario realizar para asegurar el correcto intercambio de información entre aplicaciones.

La validación de esta hipótesis pasa por realizar una comparativa entre los posibles enfoques con los que afrontar el problema (alguno de ellos ya utilizados con anterioridad) y la solución propuesta. Para ello el primer paso consiste en estudiar las opciones existentes para poder seleccionar un conjunto de ellas que refleje con claridad la situación expuesta. Una vez realizado este proceso, se ha decidido seleccionar los siguientes enfoques que forman un conjunto representativo de posibilidades de realizar intercambio de información entre aplicaciones sobre las que se podrán extraer conclusiones relevantes:

1. Cada aplicación almacena información del resto de aplicaciones existentes.
2. Cada aplicación realiza transformaciones bidireccionales en relación a un estándar.
3. Utilización de CARL como lenguaje semántico para el intercambio de datos.

El estudio del número de conversiones de datos de estos tres enfoques permitirá arrojar luz acerca de la complejidad de cada uno de ellos y comparar su viabilidad y eficiencia en los procesos de comunicación.

5.3.2.3.1 Comunicación Aplicación-Aplicación

En este enfoque se presenta la situación en la que cada aplicación posee información relativa al formato de los datos de todas las demás aplicaciones. Mediante este conocimiento, la aplicación puede consultar qué tipo de datos espera y desea una segunda aplicación y realizar la conversión a ese formato por sí mismo.

Si bien el enfoque es sencillo, resulta completamente inviable. Son muchos los problemas que impiden la implementación de este modelo. Estos problemas van desde la gran capacidad de almacenamiento que sería necesario para poder guardar la información de todas las demás aplicaciones existentes (además de los de la propia), a la dificultad que conlleva tener que almacenar campos nuevos cada vez que surge una nueva aplicación, desarrollar código específico que permita hacer las conversiones y por último el número de posibles transformaciones de datos que se podrían dar. Por todo ello nos encontramos ante un problema n-n, que indica que para cada una de las aplicaciones es necesario establecer relaciones con todas las demás, algo que como ya se ha mencionado es completamente inviable para estos sistemas de información.

La Figura 62 muestra un ejemplo del funcionamiento que tendría un sistema que utilizase este enfoque:

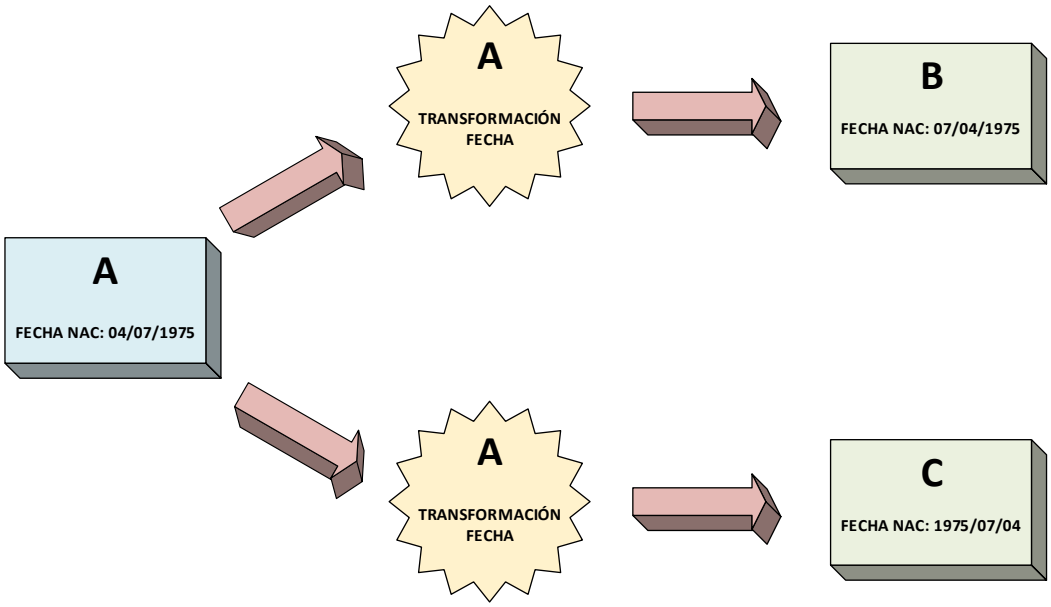


Figura 62. Proceso de comunicación Aplicación-Aplicación

Como se puede comprobar, la aplicación A debe enviar una fecha de nacimiento a las aplicaciones B y C. Las tres aplicaciones poseen formatos de fechas diferentes con lo que es necesario realizar las conversiones de datos pertinentes para adaptar los formatos y que estos sean compatibles. Mediante el enfoque que plantea esta opción, la aplicación A tendría el conocimiento relativo a los formatos que precisan B y C con lo que la propia aplicación debería tener su propio código dedicado a realizar estas transformaciones. De esta manera A realiza las conversiones mientras que B y C reciben los datos como desean.

Centrando el problema en el número de transformaciones a realizar es posible afirmar que se obtienen de realizar la siguiente operación:

*Número de transformaciones A-A= n*n*y*

Siendo "n" el número de aplicaciones existentes e "y" los campos de cada una de esas aplicaciones. Como cada aplicación debe poder hacer conversiones hacia todas las demás ("n" aplicaciones deben poder comunicarse con "n" aplicaciones) y cada aplicación posee "y" campos susceptibles de ser transformados, la operación resultante resulta de una complejidad cuadrática, o en términos matemáticos, del orden de complejidad n^2 ($O(n^2)$).

El resultado de esta operación prueba la escasa viabilidad del enfoque, ya que el número de aplicaciones puede ser muy elevado, lo que aumentaría sobremanera tanto la dificultad de gestión de la información como el tiempo y el coste necesario para ello. Además de todo esto, se deja del lado del propietario de la aplicación todas las modificaciones pertinentes hecho que puede resultar crítico aplicado al entorno de la investigación debido a que resulta un aspecto fundamental que los propietarios deseen subir sus aplicaciones a la estructura de Cloud Computing y disfruten así de las ventajas ofrecidas en lugar de incurrir en mayor número de costes y mantenimiento continuo.

5.3.2.3.2 Comunicación Aplicación-Estándar-Aplicación

La opción que se presenta a continuación tiene varias similitudes con la detallada anteriormente, pero mejora sensiblemente el número de transformaciones necesarias en el sistema. Este enfoque precisa de un estándar que marca el formato en el que se pueden intercambiar los datos. De esta manera cada aplicación sólo necesita conocer el estándar para realizar la transformación de sus datos al mismo y poder así intercambiar la información deseada. Sin embargo el proceso de transformación es doble, ya que debe ser capaz de transformar los datos desde su formato al estándar cuando envíe información y del estándar a su formato particular cuando la reciba.

La mejora con respecto al caso anterior es evidente y aunque se sigue dejando la implementación de las conversiones del lado de la aplicación, éstas son mucho menores al tener que conocer únicamente un formato al que enviar y del que recibir la información lo que reduce sensiblemente la complejidad.

La Figura 63 muestra el mismo ejemplo del caso anterior aplicado al modelo de comunicación aquí comentado:

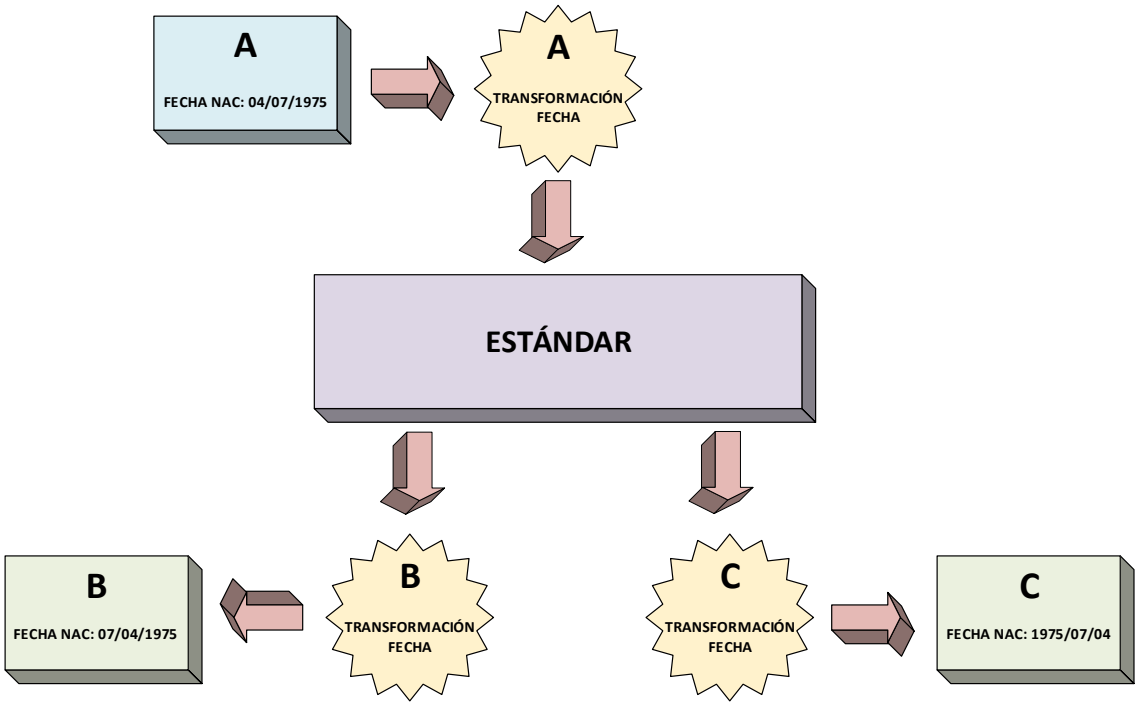


Figura 63. Proceso de comunicación Aplicación-Estándar-Aplicación

El ejemplo muestra de nuevo la situación en la que A desea enviar una fecha de nacimiento a B y C. Como se puede observar, para realizar la transformación empleando este modelo la aplicación A debe realizar la conversión de su formato al estándar. A partir de aquí, las aplicaciones B y C reciben esa fecha en el formato correspondiente al estándar y ellas mismas realizan la conversión de formato que necesitan. La diferencia con respecto al caso anterior es que en este modelo tanto las aplicaciones que envían como las que reciben deben realizar transformaciones en los datos mientras que en el caso anterior sólo precisa de estas operaciones la aplicación emisora de información. A pesar de esto, las conversiones de datos que deben hacer son menores ya que para que se produzca el intercambio no es necesario conocer el formato que necesita el resto de las aplicaciones, basta con conocer un único formato que marca el estándar. Todo ello reduce la complejidad de las operaciones y sobre todo el coste de la implementación por parte de las aplicaciones, que ahora sólo precisan un tipo de transformaciones que no variará con cada aplicación con la que se desee intercambiar información, permaneciendo así constante y evitando crear nuevo código con la aparición de nuevas aplicaciones.

De forma análoga al caso anterior, se puede calcular el número de transformaciones que serían necesarias para que se produzca el intercambio de información utilizando este segundo modelo:

*Número de transformaciones A-E-A= n*y*2*

Siendo nuevamente "n" el número de aplicaciones existentes e "y" el número de campos que posee cada aplicación. El número de conversiones se obtiene al analizar que para enviar datos cada una de las "n" aplicaciones debe ser capaz de convertir sus "y" campos al formato del estándar a la hora de enviar, pero también el proceso inverso a la hora de recibir, de ahí la multiplicación por dos, o dicho de otra manera, cada aplicación debe poder transformar sus "y" campos de su formato al estándar y viceversa. Con ello se consigue reducir la complejidad a 2n, orden aceptable para probar la viabilidad de este enfoque en entornos reales.

Aún con todo esto, sigue existiendo el problema de dejar del lado de la aplicación las transformaciones a realizar, que si bien se ha reducido con respecto al caso anterior, en aplicaciones con gran cantidad de campos pueden tener un impacto significativo para los propietarios en términos de costes y tiempo.

5.3.2.3.3 Comunicación mediante CARL

Esta última opción presenta el enfoque de CARL tal y como se ha detallado a lo largo de todo el documento. En este caso, las aplicaciones no necesitan tener ningún tipo de conocimiento ni realizar ningún tipo de conversión, ya que será el sistema del modelo CARL el encargado de hacerlo. La ontología posee la información relativa a las aplicaciones y es a través de este conocimiento donde se establece el nexo de unión entre las aplicaciones, permitiendo aislar a éstas de las incompatibilidades de los formatos y de las transformaciones.

La principal mejora que aporta este modelo con respecto al anterior es la de liberar a los propietarios de las aplicaciones del código de conversión. Esto facilita su labor y aumenta las posibilidades de que deseen subir sus aplicaciones a la nube. El número de conversiones a realizar se consigue al estudiar el ejemplo de la Figura 64:

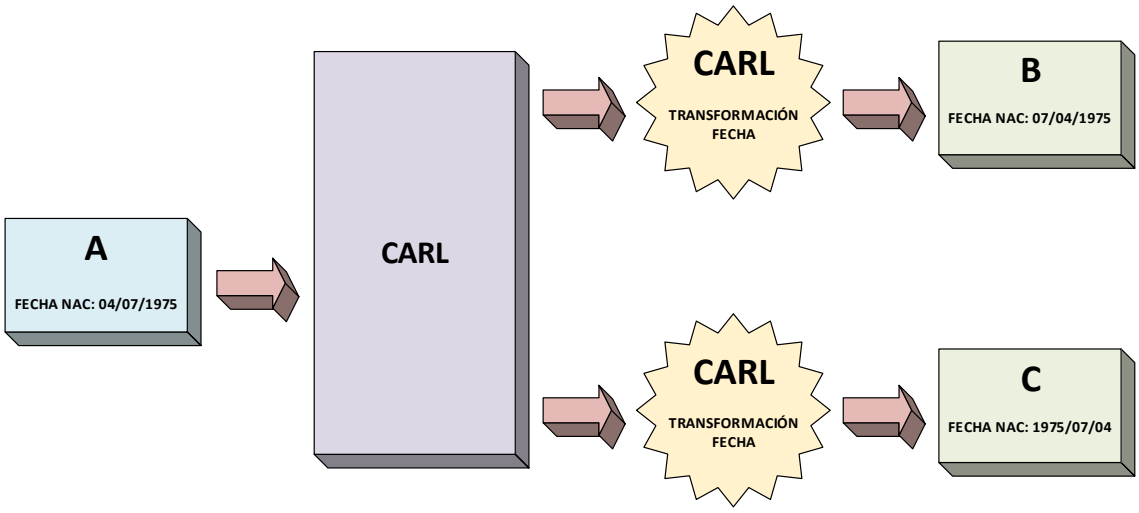


Figura 64. Proceso de comunicación mediante CARL

En este caso, cuando se quiere obtener un dato de la aplicación A, la fecha de nacimiento, las otras aplicaciones envían la petición. El sistema CARL conoce el formato en el que llega y en el que debe enviarlo y es él el encargado de hacer las conversiones necesarias en los formatos para que llegue correctamente al destino. Como se comentaba, las aplicaciones quedan fuera de todos los procesos de conversión.

Resulta evidente que las funciones de conversión serán implementadas en el sistema, que al conocer los posibles formatos en los que le pueden llegar reducen el número de posibilidades considerablemente. Por ello, para este caso el número de transformaciones viene dado por la siguiente fórmula:

$$\textit{Número de transformaciones CARL} = n*y$$

Donde "n" es el número de aplicaciones existente e "y" el número de campos en cada aplicación. En cada proceso de intercambio de información, CARL debe ser capaz de convertir los formatos de todos los campos de una aplicación en los correspondientes en cada caso. De ahí se extrae que el orden de complejidad es lineal, O(n), reduciendo los dos casos anteriormente expuestos. Además este número proporciona el máximo posible de conversiones que en pocos casos se dará ya que encontrándose los tipos de formatos limitados, muchos de ellos se repetirán y no será necesario realizar conversión alguna.

5.3.2.3.4 Discusión

Como se puede comprobar examinando los resultados expuestos con anterioridad, el enfoque de CARL supera a los demás expuestos en cuanto al número de transformaciones a realizar se refiere, aspecto principal de estudio de esta hipótesis. CARL reduce el orden de complejidad del sistema completo a complejidad lineal, probando ser más eficiente en estos términos que el resto de opciones. El gráfico de la Figura 65 muestra una comparativa del número de transformaciones que se podrían llegar a realizar en los modelos presentados con anterioridad:

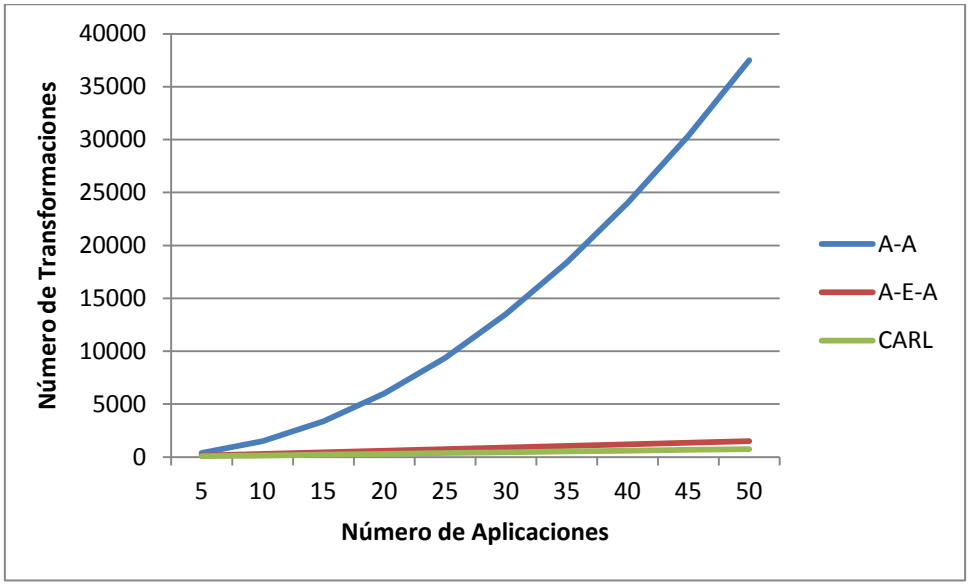


Figura 65. Comparativa del número de transformaciones en función de cada modelo de intercambio

Se puede observar como con el modelo de CARL el número de transformaciones a realizar es sensiblemente menor, con diferencias más que notables con el primer modelo (Aplicación-Aplicación) y significativas con el segundo (Aplicación-Estándar-Aplicación). Además en ambos casos las diferencias se irán incrementando a medida que el sistema vaya creciendo, ya que en este tipo de sistemas una muestra de 50 aplicaciones (como la que se muestra en el gráfico) se puede considerar como una muestra pequeña. Recordar también, que todos los valores reflejados en la gráfica son máximos, pero que en el caso de CARL, al encontrarse limitados los formatos que pueden tomar los campos de las aplicaciones en el sistema, las posibilidades de tener que efectuar conversiones son menores, lo que con casi total seguridad reducirá aún más el valor absoluto del número final de conversiones a realizar.

Como ventaja adicional, CARL libera a las aplicaciones de la implementación de funciones de conversión, imprescindibles en los otros modelos, haciendo la plataforma más atractiva para los propietarios de aplicaciones y aumentando así las posibilidades de que crezca el número de aplicaciones existentes en el sistema.

5.3.3 **Hipótesis 3**

Finalmente, la hipótesis principal que forma el último bloque enuncia que *es posible que varias aplicaciones heterogéneas intercambien información en entornos cerrados como el proporcionado por una estructura de computación en nube*, lo que centra los esfuerzos de esta hipótesis en demostrar la importancia de los entornos cerrados/limitados para poder asegurar un correcto intercambio de información entre aplicaciones heterogéneas.

Al focalizarse en este ámbito se hace importante demostrar por un lado que el intercambio de información es posible gracias a las limitaciones impuestas en el entorno y que marca una diferencia notable la ausencia de estas limitaciones y por otro, que los entornos de Cloud Computing cumplen con los requisitos y propiedades necesarios para asegurar un correcto intercambio de información.

Las siguientes hipótesis secundarias reflejan estos aspectos de interés y permiten validar la adecuación de la propuesta.

5.3.3.1 Hipótesis 3.1

En esta hipótesis se afirma que *un entorno cerrado permite el intercambio de información entre aplicaciones heterogéneas*. A lo largo del documento se ha explicado la importancia del concepto de "jardín vallado" para el desarrollo de la investigación. Esta idea resulta una de las centrales que permite demostrar que los entornos cerrados (o limitados) son necesarios para poder asegurar la interconexión entre aplicaciones heterogéneas. Para poder llegar a conclusiones fundamentadas, se realizará un análisis teórico del problema que permitirá probar la hipótesis planteada.

Como se comentaba, esta hipótesis se enfrenta al extendido problema de las comparaciones existentes entre los conceptos de mundo abierto y mundo cerrado. Entendemos por mundo abierto aquél en el que no existe limitación alguna. Situando esta definición en el ámbito propio de esta investigación, sería un mundo en el que aplicaciones completamente heterogéneas operan sin ningún tipo de restricción a nivel de datos. Planteada esta situación, se desea que dos aplicaciones intercambien información. Al estar en un mundo abierto, cada una de las aplicaciones puede tener sus datos en los formatos que ellas deseen al quedar eliminadas todas las restricciones posibles. En este punto se plantean varias opciones para el intercambio:

1. Las aplicaciones transforman el formato de sus datos para que coincidan con los de la aplicación con los que se quiera comunicar.
2. Se comunican a través de un estándar creado de manera independiente.
3. Se utiliza un lenguaje similar a CARL pero carente de restricciones de ningún tipo.

A continuación se explican en profundidad cada una de estas opciones que permitirán conocer los problemas existentes para el intercambio de información en un mundo abierto.

1. Esta situación es similar a la planteada en el apartado 5.3.2.3.1. Para que una aplicación pueda intercambiar datos con otra en este escenario sería necesario que cada aplicación tuviera información relativa a todos los formatos todas las aplicaciones existentes. Este hecho ya resulta un problema en entornos limitados por factores como la complejidad computacional, la necesidad de actualizaciones continuas por la continua aparición de nuevas aplicaciones, aumento de los costes asociados con el mantenimiento, etc. Pero todos estos problemas aumentan enormemente si lo que se desea es aplicarlo a un ámbito global, como puede ser el deseo de comunicar todos los sistemas existentes a través de Internet. Esta situación es completamente inviable por motivos innumerables y evidentes (continua aparición de nuevas aplicaciones, costes de mantenimiento insostenibles, mayor tiempo de desarrollo que de explotación, etc.). La Figura 66 refleja gráficamente esta situación en la que una aplicación A debe hacer tantas transformaciones de datos como campos haya por cada aplicación, originando una complejidad de orden cuadrático y haciendo imposible su implementación.

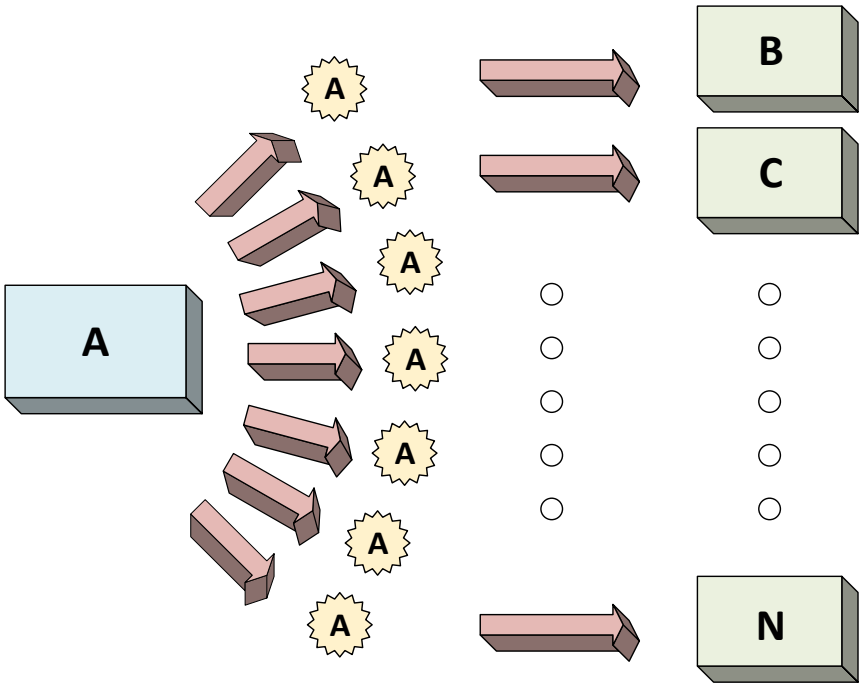


Figura 66. Modelo de comunicación entre aplicaciones en mundo abierto

2. En esta situación aparece la figura de un estándar de comunicación que fija una serie de formatos para el intercambio de información, similar al caso planteado en el apartado 5.3.2.3.2. En ese ejemplo se comenta la posibilidad de que las aplicaciones posean información referente al estándar, traduzcan sus formatos al marcado por él y de esa manera puedan enviar y recibir información que ellos mismos conviertan. Esta situación parece ser aplicable sin establecer restricciones, en un mundo abierto, pero no es así. Si se trabaja en un mundo abierto es más que probable que el estándar no cubra todos los posibles campos que reflejen las aplicaciones. Es importante entender que sin limitación cualquier cosa es factible, por ejemplo, puede existir una aplicación que precise de un campo número de serie y no exista una correspondencia para él en el estándar, u otros en los que el estándar sea demasiado restrictivo, puede fijarse la longitud máxima para un número de socio en 15 caracteres, pero una aplicación precisar de un número de socio de 100 caracteres. Al estar en un mundo abierto cada uno pone sus reglas para su aplicación, creando un pequeño caos cuando quieren intercambiar información entre ellas.
   Además de esto existe el problema añadido de dejar estas acciones del lado de cada aplicación, en el que el desarrollador debe conocer el estándar y además realizar correspondencias entre sus campos y los del estándar ya que, el campo “Autor” en el estándar puede corresponderse con el campo “NombreAutor” de la aplicación y no la aplicación no entiende esta relación salvo que el propietario de la misma realice las conversiones a ese formato. La Figura 67 ilustra los problemas aquí mencionados:

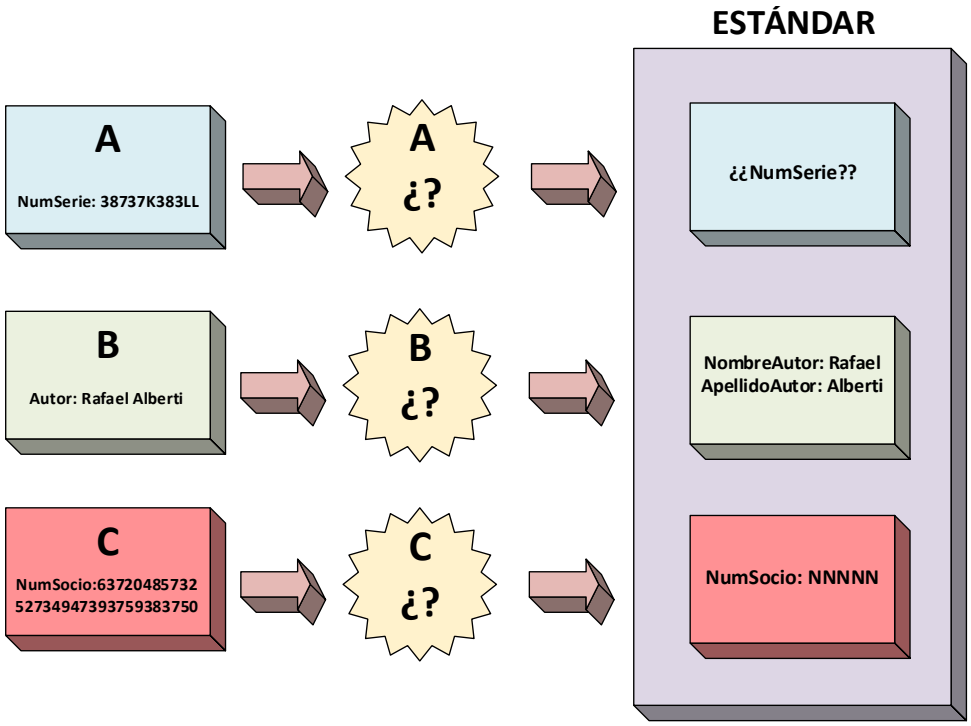


Figura 67. Modelo de comunicación mediante estándar en mundo abierto

En el ejemplo se observa que la aplicación A no encuentra el campo que necesita traducir en el estándar, la aplicación B no encuentra una correspondencia en cuanto al nombre del campo y su estructura y la aplicación C no puede representar el valor de su número de socio en el formato estándar ya que éste limita demasiado el número de caracteres válidos. Todos estos son algunos de los problemas que impiden la implementación de este modelo en el mundo real ya que o el estándar se encuentra actualizándose continuamente con los campos que necesitan las aplicaciones que surjan (aspecto que nos lleva al problema anterior, 1) con la consiguiente actualización de las aplicaciones para adaptarse a ese nuevo estándar o habrá muchas aplicaciones que queden excluidas por no poder representar los campos que necesitan para su funcionamiento.

3. Esta opción afronta el enfoque de la utilización de un lenguaje similar a CARL pero en el que no existan restricciones en cuanto al nombre y tipo de los datos. El potencial de CARL reside en sus propiedades semánticas y en el almacenamiento de todo el conocimiento relativo a las aplicaciones existentes en el jardín vallado. Al eliminar esto último el sistema pierde el control relacionado con las aplicaciones y hace imposible los procesos de intercambio de información.

   En primer lugar, sin limitaciones, los campos de cada aplicación no estarían obligados a tener una correspondencia con los campos de la ontología del dominio planteada. Esto impediría al lenguaje establecer relaciones entre los campos de las aplicaciones con nombres diferentes, es decir, no sabría que lo que una aplicación representa como “FechaNacimiento” y otra como “FNac” son conceptualmente iguales, permitiendo además la redundancia en los datos. Otro problema surge con los formatos. CARL permite una serie de formatos diferentes en aquellos campos susceptibles de tener varios igualmente válidos, si bien, en un mundo abierto cualquier aplicación puede representar los datos libremente, con lo que, si se desea puede, por ejemplo, puede representar un DNI de forma inversa y el lenguaje no tendría la posibilidad de crear una función de conversión apropiada. La Figura 68 muestra los principales problemas de este modelo de una forma gráfica:

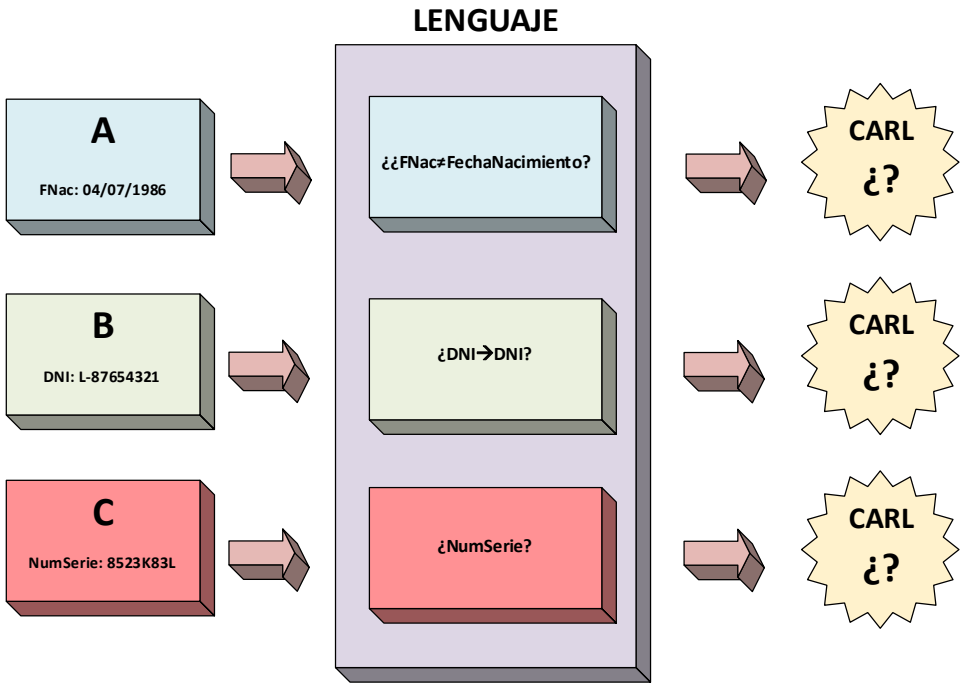


Figura 68. Modelo de comunicación mediante lenguaje en mundo abierto

Como se puede comprobar en la figura se muestran los problemas mencionados. La aplicación A desea intercambiar su fecha de nacimiento pero no entiende que el campo "FNac" se corresponde con ello, la aplicación B almacena el DNI de una forma específica y CARL no sabe como transformar ese DNI en un formato adecuado para el destino y por último la aplicación C quiere enviar un número de serie sin embargo ese campo no existe en la ontología y por lo tanto no posee correspondencia ni función de conversión.

Como se puede comprobar, los problemas que surgen en un mundo abierto son numerosos y en su mayoría difíciles de solucionar sin el establecimiento de unas restricciones que permitan crear un entorno más reducido y controlado.

5.3.3.1.1 Discusión

Los conflictos entre el mundo abierto y el mundo cerrado se han dado en diferentes ámbitos, disciplinas y problemas. La discusión entre ambos enfoques siempre ha suscitado polémica y las ventajas e inconvenientes de ambos hacen que sean adecuados o no según el enfoque particular del problema.

En el ámbito en el que se desarrolla esta investigación se ha podido comprobar la necesidad de establecer una serie de restricciones y límites (el denominado "jardín vallado") que permita conocer la información referente a las aplicaciones existentes en la plataforma de Cloud Computing para así poder establecer relaciones entre ellos. Estos nexos de unión se crean gracias a las limitaciones proporcionadas por la ontología del dominio y por la obligación de las aplicaciones de aceptar el hecho de proporcionar la información referente a los campos, sus formatos y su correspondencia en GoodRelations. Se puede pensar que con este planteamiento se genera un entorno en ocasiones demasiado limitado, pero es precisamente éste uno de los objetivos de la investigación, limitar las propiedades para que las aplicaciones que acepten las condiciones puedan interoperar entre sí automáticamente, accediendo a una serie de ventajas que compense el ceder a esas exigencias.

Además de todos los argumentos que muestran la imposibilidad de alcanzar interoperabilidad sin unas restricciones claras y definidas, la Historia ha mostrado como las propuestas de solución demasiado amplias o generales han fracasado en su intento por crear un mundo completamente interoperable (WSMO) debido en gran medida a los impedimentos mostrados a lo largo de este apartado. Por todo ello, la conclusión general que se extrae de este estudio es que se ha demostrado que los entornos cerrados permiten el intercambio de información entre aplicaciones gracias a las limitaciones a las que éstas se adaptan que consiguen proporcionar el conocimiento suficiente al sistema para favorecer los intercambios de información solucionando las incompatibilidades que se sucedan de forma automática, operaciones completamente inalcanzables en entornos abiertos en el que cada uno establece sus propias reglas.

5.3.3.2 Hipótesis 3.2

La hipótesis objeto de este apartado postula que *los entornos de Cloud Computing soportan la interacción automática de las aplicaciones alojadas en ellos*. Para su demostración se empleará un doble enfoque basado en aspectos teóricos y en pruebas realizadas mediante el software desarrollado para ello.

Dado que el Cloud Computing es un paradigma bastante reciente a pesar de su gran crecimiento en los últimos años, los trabajos existentes en el campo son limitados y de reciente creación. Aun así, existen algunos sistemas que demuestran la posibilidad de realizar interacciones automáticas en estos entornos. Un ejemplo de ello se encuentra en (Zhou, Fang, & Chen, 2012), un sistema para descubrimiento y selección de servicios y paquetes de viajes basado en OWL y BPEL y pensado para entornos de computación en nube. Gracias al uso de las tecnologías semánticas (en este caso OWL-S entre otros), se permite la interacción automática entre servicios/aplicaciones de viajes a través de servicios web con el objetivo final de obtener la oferta más adaptada a las necesidades del usuario. Este trabajo se encuentra en fase prototípica, aunque los resultados iniciales confirman que las interacciones automáticas son posibles. Como se puede comprobar, las similitudes de este trabajo con la investigación presentada son considerables, ya que hacen uso de la semántica y los entornos cloud para conseguir sus objetivos.

Existen otros enfoques que persiguen los objetivos de interoperabilidad en diferentes campos, pero que se sirven de los entornos de Cloud Computing para conseguirlo y que demuestran la adecuación del paradigma para lograrlo. Tal es el caso de (Casola, Rak, & Villano, 2010) que utiliza el IaaS para lograr la interoperabilidad con sistemas basados en grid, o el caso de (Pastor, Giachetti, Marín, & Valverde, 2013), más centrado en el campo de la ingeniería del software que persigue la automatización de la interoperabilidad de modelos conceptuales en entornos como los de computación en nube y por último el trabajo presentado por (Khriyenko, Terziyan, & Kaikova, 2013) que se podría definir como más futurista y que no se encuentra tan centrado en los entornos de computación en nube sino en la perspectiva de la futura creación y establecimiento del Internet de las Cosas donde se persigue la interoperabilidad con un sistema basado en la asistencia del usuario y el ontology alignment.

Todos estos ejemplos de sistemas que persiguen los objetivos comentados y que utilizan el Cloud Computing para ello, permiten demostrar que este paradigma resulta no sólo adecuado para lograr los retos planteados, sino que en ocasiones, resulta completamente necesario como se ha demostrado en la validación de hipótesis anteriores (ver 5.3.3.1).

Una vez aportado el enfoque teórico basado en la experiencia de trabajos previos, se presenta a continuación una demostración más práctica que complementa a la anterior y prueba que el sistema diseñado habilita la interacción automática de las aplicaciones alojadas en la estructura de computación en nube. Para ello se ha utilizado el software desarrollado para la validación y que permite mostrar que las aplicaciones realizan procesos de intercambio de comunicación sin interacción específica por parte del usuario.

Para ello haremos uso de un ejemplo anterior en el que se mostraban las diferentes interacciones realizadas entre las aplicaciones de “Iberia” y “BBVA”. Previamente en la sección 5.3.1.2 de este mismo capítulo, se mostraba la secuencia de pasos a realizar entre estas dos aplicaciones para intercambiar la información deseada. Sin embargo, gracias al software desarrollado se pueden mostrar las actividades que realiza el sistema completo sin precisar la interacción del usuario. La Figura 69 refleja esta situación:

CARL > Data exchanged between Iberia and BBVA

Iberia					BBVA			
Field	**Matches**	**Type**	**Value**		**Field**	**Matches**	**Type**	**Value**
Name	GR:Individual:Name	String	Miguel	→	NombreCliente	GR:Individual:Name	String	Miguel
Identifier	GR:Individual:SerialNumber	String	47451012L	→	DNI	GR:Individual:SerialNumber	int	47451012
CuentaCliente	GR:Offering:SerialNumber	String	6372372736278273#	←	NumeroCuenta	GR:Offering:SerialNumber	int	6372372736278273

Figura 69. Intercambio automático de información entre Iberia y BBVA

Como se puede observar, la Figura 69 muestra el ejemplo comentado previamente pero en este caso se observa como las operaciones han sido realizadas directamente por el lenguaje y no de forma unidireccional. Por ello se indica qué datos se han intercambiado, en qué dirección y las transformaciones realizadas sobre ellos demostrando así la posibilidad de que los intercambios se realicen de forma automática.

Una vez mostrado esto, se puede concluir que CARL ha podido solventar los problemas de comunicación y heterogeneidad, alcanzando la interoperabilidad automática entre las aplicaciones, hecho que, junto con las observaciones teóricas proporcionadas en el inicio de la sección, confirma la hipótesis planteada.

5.4 Sumario

El proceso de validación llevado a cabo presenta un análisis detallado de cada una de las hipótesis expuestas en la sección 3.1. El estudio de cada una de ellas se ha realizado de manera independiente y basándose en diversos aspectos ya que, debido a la heterogeneidad de las mismas, se imposibilita el establecimiento de un mismo método para su demostración. De esta forma se han fijado una serie de parámetros para su examen y los enfoques más adecuados para alcanzar unos resultados fiables y concluyentes.

Como se acaba de comentar, una de las mayores dificultades al enfrentarse al proceso de validación se ha encontrado a la hora de seleccionar la forma más adecuada de afrontar la demostración de cada hipótesis. Los objetivos de esta tesis se centran en la consecución de una serie de tareas e intercambios de información, que resultan ser aspectos difícilmente cuantificables. Principalmente por ello, se ha diseñado un sistema de validación que permite mostrar algunos de los procesos más relevantes que se suceden en el sistema y que facilita la verificación de su funcionamiento. Las capturas del sistema expuestas a lo largo del proceso de evaluación y validación muestran diversos aspectos del funcionamiento de CARL y del resto de componentes con los que trabaja, que prueban que todos ellos son capaces de realizar las tareas para las que han sido diseñados y que se alcanzan los resultados esperados.

Varias de las hipótesis de la investigación poseen un carácter más racional o filosófico que hacen que su demostración se encuentre más centrada en principios teóricos relacionados con las tecnologías y conceptos que involucra en lugar de basarse en aspectos medibles o mostrables. Estas hipótesis han sido analizadas y probadas desde esta perspectiva, argumentando de forma lógica y razonada los elementos en los que se fundamentan para poder proporcionar una visión objetiva del problema y, consecuentemente, de la solución que se propone. En algunos de los casos ha sido posible apoyar las explicaciones teóricas con capturas del sistema de validación, sin embargo en otros casos ha resultado imposible debido al propio enunciado de la hipótesis.

Una vez finalizado el proceso de evaluación y validación se puede concluir que los resultados conseguidos han sido muy satisfactorios, quedando demostradas todas las hipótesis que se plantearon al inicio de esta tesis doctoral. Estos resultados confirman el éxito de la investigación y las enormes posibilidades que derivan de ella. Con todo ellos es posible afirmar que la tesis doctoral presentada realiza las contribuciones necesarias para las que fue diseñada y supone un progreso significativo en las diferentes áreas de investigación relacionadas, permitiendo avanzar en la consecución de entornos interoperables.

6 Conclusions and Future Research

In order to satisfy the requirements for the International doctorate mention, this chapter includes the English translation of the conclusions of this doctoral thesis.

In this chapter the final conclusions of the investigation performed are presented. Joint with the review of the main contributions of this doctoral thesis, new research lines are proposed as a continuation of the work carried out. Finally, at the end of the chapter, a list of publications performed in different scientific areas and within the context of this research is provided.

6.1 Conclusions

From the very beginning the motivation of the author for this research was to bring closer the domain where this doctoral thesis is focused: the interoperability between applications in Cloud Computing environments. As have been mentioned before, the wide variety of solutions is elevated, however not all the approaches allow confronting the challenges set out by the investigation and solving the complex problems exposed.

Along the document that shows the investigation done, it has gone in depth in the different areas that it covers for supplying solid and based foundations. Therefore, it has been possible to define appropriate and interesting hypotheses that have guided the whole process to the obtaining of relevant results for the research community.

The analysis of the state of the art has clarified different aspects about the technologies and concepts used during the investigation, allowing setting a starting point for initiating the contributions of the thesis.

One of the main terms that are faced during the researching is the interoperability. The problem of interoperability lies on the impossibility of two or more systems for exchanging information that they can use afterwards. This problem has been studied for decades and from different perspectives, though nowadays there is not a generally accepted solution. All of this has caused a situation where the absence of standards and the growing number of information systems have increased the difficulties. This fact make impossible to take advantage of the application´s potential and, in many cases, they become pure information repositories without the possibility of an efficient exploitation. Looking at the literature, it is possible to find previous and related works and initiatives in different ambits. In some of them great amounts of money were invested with the objective of getting a valid solution to this problem, but all of them failed. The reasons for these failures are varied, nevertheless, in the majority of the cases the problem resides in the wide scope of the proposal. They try to offer a too general or global

solution, producing again the main scenarios that created the problem. The conclusion extracted from this situation was that it is necessary to limit the environment and establish a set of prerequisites, for enclosing the problem, in order of being able to achieve interoperability.

Exist different ways of establishing the limits of action or the working environment. For the goals of this research the decision was to adopt a combined approach through Semantics and Cloud Computing platforms. In this way it has been possible to use Cloud Computing infrastructures for building that can be defined as a semantic PaaS in which is possible the exchange of data between the applications hosted in there.

Cloud Computing environments have rapidly increased its presence and importance in the sector and the predictions affirm that the growth will be even higher. This instant success is due to its all company spectrum attractive features that observe how they can reduce costs and delegate some business external tasks to the server provider. Without doubts, the paradigm of Cloud Computing has marked an inflexion point in the way we conceive software, starting from the pay-per-use payment method that breaks the traditional model based on licenses, open new skylines in this regard. For these and other reasons, Cloud Computing has become the ideal environment for the study of interoperability. Dealing with an environment that can be considered “limited”, it is possible to establish some requirements that have to be fulfilled for being able of accessing it, limiting at a first stage the scope of the work and setting up what is denominated as “walled garden”. Additionally, this paradigm needs an identification of all kind of users who have to be invoiced and who needs obtain permissions depending on their levels and the applications they want to subscribe. With this, it is possible to monitor and know the data of users and their applications that at last stage allow the interoperability processes. Besides, being possible to establish these requirements and recognise the user data, the same principle can be applied to the applications that want to be in the Cloud Computing platform. Therefore, when an application wants to be placed in the infrastructure, it will be necessary to accept a set of prerequisites that allow knowing information about it. This data will be stored for make use of them when necessary. Studied this option in depth, it is possible to say that for its current interest awaken, the growth and presence in the sector, performance and properties, this paradigm become suitable for the achievement of the outlined aims.

As has been previously commented, the proposed solution is based in the use of semantic technologies because they are the most appropriate ones for taking advantage of the information collected from users and applications in the Cloud Computing environment suggested. Inside the Semantic Web, the ontologies gather special importance for the development of the work. Thanks to these tools it has been possible to store the information coming from applications and establish a set of relationships between them which have made these data will be transformed in ready to use knowledge. These ontologies serve as a base for the developed language, CARL, whose functioning is also based in the concepts structured in the ontology. The Semantic Web is one of the disciplines that has strongly worked around this problem since its beginning, fact which has made a great influence in the choice of it use for the investigation captured in this document. The contribution of Semantics on the achievement for interoperability through the years is incalculable and it is possible to affirm

that, in the scope of this research, the exposed results have not been reached without its contributions at different levels.

The choice of Cloud Computing as environment and semantic technologies like the main tools for achieving an interoperable system, lack any value themselves. It is necessary an element able to use the knowledge stored in the ontology and to determine the actions that should be done in each moment for solving the incompatibilities merged at data level and guide the exchange information processes. From this side came up the idea of creating a language that allow establishing communicative processes between heterogeneous entities. The semantic language built has been able to manage the communication between applications, solving their heterogeneity thanks to well-defined data transformation functions and also to the walled garden provided by the cloud platform. For this reason, a minimum syntax were defined for developing the necessary functionality and adopting a semantic based on description logics which allow inference processes over the data for extracting the necessary knowledge in each case. In this way, and with the aid of web services as communication tools, CARL has know the needs of the applications and provides a solution in an automatic way, supposing a great advance compared with the previous related work.

For ensuring that CARL could perform its functioning, the design of a system around it has been necessary. This system will be in charge of managing supplementary elements participants in the communication processes. For all of them, a preliminary analysis of alternatives were done, allowing a decision making process for the choice of the best option. This is the case of the three layered architecture design, the database design for storing user's information and the use of the GoodRelations vocabulary as domain ontology, which has become a great importance entity for the development of CARL.

Other supplementary modules with special relevance for providing the interoperability through CARL are the figures of the PaaS Application Engine and the Interpreter. The interpreter is in charge of establishing the equivalences between CARL and the semantic of OWL-DL that allow using description logic reasoners. On the other hand, the engine manages the communication processes jointly with CARL and makes also the transformation functions when necessary for adapting the data formats of the applications. As it is easy to see, the contribution of these entities is crucial and both are considered as irreplaceable for reaching the outlined goals of the research.

The process of validation of the different properties of the language and system proposed has demonstrated the exposed hypotheses, proving the viability and effectiveness of the solution. With this procedure has been confirmed, among other, that an ontology based semantic language is a valid solution for the interoperability problem, that the environments provided by the Cloud Computing paradigm allow creating limited environments where some prerequisites can be established and that thanks to semantic technologies it is possible to automate information exchange processes. With the obtained results it is possible to conclude that the proposed solution has accomplish the goals of the investigation and also make some interest contributions for the sector and for the research community, presenting an innovative and efficient solution to the problems being studied.

The development of this doctoral thesis has taken a step forward in the studied research lines and has entailed and organized, exhaustive and careful work. For this reason, some problems have appeared during the investigation and should be mentioned because the process of solving them has contributed to increase the quality of the work performed. First of all, the lack of awareness in some of the topics for the author made that the study of the state of the art was too wide when analyzing technologies and possible approaches to face the objectives of the research with. This fact made difficult the first stages of a work complicated as it is due to the different areas that cover. On the other side, the creation and design of a language is always a difficult task due to the necessary initial definition of functionality and the prerequisites which have to be fulfilled and that have to be considered for its syntax and semantic. These actions raise the complexity of the language design which, in addition, must interact with other elements in the same environment, allowing logical inference processes. Finally, other problem merged was the configuration of the validation methodology. The goals of the research are focused on the design of a language that permits the information exchange between applications, goals that are difficult to quantify because they are not based in numerical parameters. Due to this, it was decided to design a system which allows showing that the operations carried out were correctly done and that the different environments, technologies and concepts used were contributing to the achievement of those goals. As have been mentioned before, these difficulties found along the research have caused a qualitative leap that has been reflected in the obtained results and that has allowed to advance in a direction of great interest for the research community. In addition, the conclusions extracted from this doctoral thesis can motivate the continuation of works in this research lines, making parallel investigations in the same areas.

As a final conclusion, it is possible to affirm that with this research the determined goals have been accomplished and the proposed solution suppose a valid approximation for facing the interoperability problem between heterogeneous applications at data level. The presented report reflects the results of an arduous research that has covered a wide state of the art in the subjects of relevance for the development of the work, the statement of goals and hypotheses that have guided the investigation process, semantic language design aspects and also for the system where it is framed and finally the evaluation and validation of the hypotheses which has demonstrated each of the proposed axioms. For all of this, it is possible to conclude that this doctoral thesis is successful and suppose an interest contribution in the areas related to the research performed.

Finally and a personal assessment, I just want to say that the development of this doctoral thesis has been a more enriching experience that I could ever imagine. Even though it is true that is certainly a complicated and hard work, it has made me grow up as a professional and as a person. In all facets of my life, one of my personal goals is to be proud of the things I do. Once this work is concluded I can say that I have accomplished it. After a very long time dedicate to the elaboration of this research, I think that we have put our two cents in the areas studied making our small contributions to them. In spite of this, it is only that. There is much more to do. I wish that this was the beginning of a large research career that allow me continue working on this matter, shed light on problems and get closer utopias like the achievement of a completely interoperable world.

6.2 Future Research

The research performed presents a solution for the interoperability problem based on the creation and design of a language based on semantic technologies. Although it is true that the results obtained in this doctoral thesis have been satisfactory, there are some aspects that remain still open dealing with alternative solutions, possible improvements and related research lines.

First of all, the proposed solution is one inside a wide variety of options. Along the document all the decisions taken for reaching the final solution have been justified. These decisions processes have been made based on objective aspects but also based on a set of concepts considered as priority interest. If the preferences change, new ways of exploring open. Each of them turns out to be a new line to investigate that can show similar results to the ones exposed in this research or otherwise obtain relevant results in other directions.

Another aspect for continuing with the investigation is the language design itself. One of the desired goals was that the language had the minimum syntax definition for managing the communication between applications and their associate processes. In this line it would be possible to add and modify a lot of aspects of the language, also a semantic and syntax level, which provide a new dimension to the solution. Inside this ambit, the knowledge representation model defined in the ontology for describing the different concepts and relationships also opens a wide spectrum of future research lines.

One aspect that would provide more versatility to the system and that could bring a greater amount of application and users would be the increase of the domain ontology vocabulary. This way, the number of restrictions at field level will be softened, allowing more flexibility without losing control of the information stored in the system. Nevertheless, and has been commented along the document, this modification involve a higher development cost because it will be necessary the implementation of a greater number of transformation functions which take into account the new inserted fields. Other option consider in this line was offer to the users the possibility of create new fields or request their creation to the administrator of the system. The problem arises was exactly the same. If fields can be created by the tenants, the process of transformation function creation become very complicated, besides there will be a lapse of time when will be impossible to offer interoperability for this new field. On the other hand, offering the possibility of requesting the creation of the field to the administrator, the process will be slowed down, but this is compensate for security aspects due to the administrator can add this new aspect directly to the vocabulary, being structured from the very first time. In any case, these aspects would stay for the study in future investigations.

Precisely dealing with security, many possibilities exist. For the work performed these aspects have been avoided for consider them out of the scope of the research. Nevertheless, in some stages some considerations brought up for future extensions in this sense. The first of them was the option of define determined sensitive information (economical, medical, etc.) as confidential for avoiding applications accessing it directly from the ontology and following an

alternative path that provides more security and privacy to the users. This supposes also an extension on the language itself, in the design of the ontology and y the global functioning of the system which have to take into account the new established requirements. In this sense there are several alternatives that stay still open as a continuation of the work performed and that suppose interesting lines for future investigations.

Due to the conclusions extracted from the study, there is a great open research line for applying the exposed principles on this doctoral thesis to other environments where the achievement of interoperability would be necessary, that is to say, not so clearly oriented to services and applications. It is true that Semantic Web has been dedicating for years to add metadata to contents for facilitating these processes, but its use in this research can open new horizons in this ambit.

As a final point, it has to be mentioned that probably one of the most interesting lines to work in is in trying to reach interoperability between different Cloud Computing environments. Theoretically and following the principles exposed in this investigation, it is presented as something viable but, without a doubt it requires a set of meticulous studies for being able to sketch a valid solution to the problem. In spite of the existence of a great number of probabilities, this is not a trivial task due to the work will face the problems of global solutions that historically has made fail the previous attempts in this line. However, if achieved, the possibilities will be enormous. It will be possible to obtain the same benefits that are presented in this doctoral thesis but at a bigger scale, supposing a great advance in the achievement of an interoperable world.

6.3 Publications

As a consequence of this investigation, the following publications have been conducted in different scientific areas:

- **CARL: A Complex Applications Interoperability Language based on Semantic Technologies for Platform-as-a-Service Integration and Cloud Computing**. E. Jiménez-Domingo, J.M. Gómez-Berbís, R. Colomo-Palacios, Á. García-Crespo. *Journal of Research and Practice in Information Technology, vol. 43 (3) (2012).*

- **Improving Accuracy of Decision Trees Using Clustering Techniques**. Javier Torres Niño, Alejandro Rodríguez González, Ricardo Colomo Palacios, Enrique Jiménez-Domingo, Giner Alor-Hernández. *J. UCS 19(4): 484-501 (2013)*

- **AKNOBAS: A knowledge-based segmentation recommender system based on intelligent data mining techniques**. Alejandro Rodríguez González, Javier Torres Niño, Enrique Jiménez Domingo, Juan Miguel Gómez Berbís, Giner Alor Hernández. *Comput. Sci. Inf. Syst. 9(2): 713-740 (2012)*

- **Using agents to parallelize a medical reasoning system based on ontologies and description logics as an application case**. Alejandro Rodríguez González, Javier Torres

Niño, Gandhi Hernández-Chan, Enrique Jiménez-Domingo, José María Álvarez-Rodríguez. *Expert Syst. Appl. 39(18): 13085-13092 (2012)*

- **Using Ontologies in drug prescription: The SemMed approach**. A. Rodríguez-González, Á. García-Crespo, R. Colomo-Palacios, J. M. Gómez-Berbís, E. Jiménez Domingo. *International Journal of Knowledge-Based Organizations, vol. 1 (4), 2011.*

- **Multi-Tenancy: A new architecture for clouds** (Book Chapter). E. Jiménez-Domingo, A. Lagares-Lemos, J. M. Gómez Berbís, in *Cloud computing: methodology, system, and applications, CRC Press (2011).*

- **CLOUDIO. A Cloud Computing-oriented Multi-Tenant Architecture for Business Information Systems**. E. Jiménez Domingo, J. Torres Niño, Á. Lagares Lemos, M. Lagares Lemos, J. M. Gómez Berbís, R. Colomo Palacios. *IEEE Cloud 2010, Miami, USA, July 5-10. 2010.*

7 Conclusiones y Líneas Futuras

Con el fin de cumplir con los requisitos para la obtención de la mención internacional del doctorado, este capítulo sobre las conclusiones y trabajos futuros fue escrito originalmente en inglés. A continuación se incluye la traducción del mismo a la lengua española.

En este capítulo se presentan las conclusiones finales de la investigación expuesta en la esta tesis doctoral. Junto con el repaso de las principales aportaciones se proponen un conjunto de futuras líneas de investigación como continuación de la misma. Para finalizar se enumeran las publicaciones efectuadas a raíz del trabajo realizado en diferentes ámbitos científicos.

7.1 Conclusiones

Como punto de partida para este trabajo se ha querido aproximar al lector al dominio en el que se centra esta tesis doctoral: la interoperabilidad entre aplicaciones en entornos de Cloud Computing. Como se ha podido comprobar, el abanico de soluciones posibles es muy elevado y variado, sin embargo, no todos los enfoques permiten afrontar los retos planteados por la investigación y solucionar los complejos problemas que se presentan.

A lo largo de todo el documento que acredita y expone la investigación realizada, se ha profundizado en las distintas áreas que se abarcan para dotar al trabajo de una base sólida y fundamentada. De esta forma se han podido definir unas hipótesis adecuadas e interesantes que han guiado la totalidad del proceso hacia la obtención de unos resultados relevantes para la comunidad investigadora.

El análisis del estado del arte ha clarificado diferentes aspectos sobre las tecnologías y conceptos utilizados en la investigación, lo que ha permitido fijar un punto de partida sobre el que iniciar las aportaciones del trabajo.

Uno de los términos principales que se afrontan es el de la interoperabilidad. El problema de la interoperabilidad consiste en la imposibilidad de dos o más sistemas de intercambiar información para utilizarla posteriormente. Este problema se ha estudiado desde hace décadas y desde distintas perspectivas, sin embargo aún hoy no se ha encontrado una solución generalmente aceptada. Esto ha provocado una situación en la que la inexistencia de estándares y el creciente número de sistemas de información han acrecentado la problemática. Este hecho hace que no sea posible aprovechar totalmente el potencial de las aplicaciones y, que en muchos casos, se conviertan en meros repositorios de información sin

posibilidad de explotación de forma eficiente. Observando la literatura se han encontrado trabajos e iniciativas previas en diferentes ámbitos, en algunas de las cuales se invirtieron enormes cantidades con el fin de llegar a soluciones válidas para este problema. Todas ellas fracasaron. Las razones de estos fracasos son variadas, sin embargo, en su mayoría el problema reside en la gran amplitud de los enfoques, que pretenden dar una solución demasiado global o general, lo que reproduce los principales escenarios que en su momento crearon el problema. De esta situación se extrajo la conclusión de que resulta necesario limitar el entorno y establecer una serie de requisitos, de forma que se acote el problema, para poder conseguir la interoperabilidad.

Existen diferentes formas de establecer los límites de acción o el entorno en el que se trabaja. Para los objetivos que se querían alcanzar en la investigación se decidió un enfoque combinado a través de la semántica y las plataformas de Cloud Computing. De esta forma se han podido utilizar infraestructuras de computación en nube para construir lo que se puede definir como un PaaS semántico, en el cual se hace posible el intercambio de datos entre las aplicaciones que ahí residen.

Los entornos de Cloud Computing han incrementado rápidamente su presencia e importancia en el sector y las previsiones marcan que se producirá un crecimiento aún mayor. Este éxito instantáneo se debe a sus propiedades, que lo hacen muy atractivo para todo el espectro de empresas que observan cómo pueden reducir sus costes y delegar ciertas tareas externas al negocio en el proveedor del servicio. Sin duda, el paradigma de la computación en nube ha marcado un punto de inflexión en la forma de concebir el software, ya que el modelo de pago por uso que ofrece rompe con el modelo tradicional basado en licencias abriendo nuevos horizontes en este aspecto. Es por estas y otras razones por las que el Cloud Computing resulta un entorno ideal para el estudio de la interoperabilidad. Tratándose de un entorno que se puede denominar “cerrado”, es posible establecer ciertos requisitos que se deben cumplir para poder acceder a ella, limitando en un primer momento el ámbito de trabajo y formando lo que se ha denominado como “jardín vallado”. Adicionalmente, este paradigma precisa de una identificación de todos los tipos de usuarios a los cuales se debe facturar y dar permisos en función de sus niveles y de las aplicaciones a las que se suscriban. De esta forma se pueden monitorizar y conocer datos de los usuarios y sus aplicaciones que en último término permitan los procesos de interoperabilidad. Además, siendo posible establecer estos requisitos y conocer los datos de los usuarios, se puede aplicar el mismo principio a las aplicaciones que se deseen alojar en la plataforma de computación en nube. Así, cuando se desee situar una aplicación en la estructura, será necesario aceptar una serie de prerrequisitos que permitan conocer información de las propias aplicaciones y de este modo almacenar estos datos para hacer uso de ellos cuando se desee conseguir la interoperabilidad. Estudiada en profundidad esta opción, se puede decir que por su actualidad, crecimiento y presencia en el sector, rendimiento y propiedades, este paradigma resulta idóneo para la consecución de los objetivos planteados.

Como se comentaba previamente, la solución planteada se basa en la utilización de tecnologías semánticas, ya que resultan más que adecuadas para aprovechar al máximo la información recogida de usuarios y aplicaciones en el entorno de Cloud Computing planteado. Dentro de la Web Semántica, las ontologías cobran especial importancia para el desarrollo del trabajo. Gracias a estas herramientas se ha podido almacenar la información procedente de las

aplicaciones y establecer una serie de relaciones entre ellos que han hecho que esos datos se transformen en conocimiento utilizable. Estas ontologías sirven además como base para el lenguaje desarrollado, CARL, cuyo funcionamiento se basa en los conceptos estructurados en la ontología. La Web Semántica es una de las disciplinas que más ha trabajado desde su nacimiento en torno a este problema, hecho que ejerció gran influencia en la elección de su utilización para la investigación objeto de este documento. La aportación de la semántica en la consecución de la interoperabilidad a lo largo de los años es incalculable y se puede afirmar que, en el ámbito de este trabajo, sin su aportación no se habrían alcanzado los resultados expuestos.

La elección del Cloud Computing como entorno y de las tecnologías semánticas como herramientas para alcanzar un sistema interoperable carecían de sentido por sí solas. Es necesario un elemento capaz de utilizar el conocimiento almacenado en las ontologías y de determinar las acciones a realizar en cada momento para solventar las incompatibilidades que se produzcan a nivel de datos y guiar los procesos de intercambio de información. De ahí surge la idea de crear un lenguaje que permita establecer procesos comunicativos entre entidades heterogéneas. El lenguaje semántico creado ha sido capaz de gestionar la comunicación entre aplicaciones y de solventar su heterogeneidad gracias a funciones de transformación de datos bien definidas y también al jardín vallado proporcionado por la plataforma cloud. Para ello se definió una mínima sintaxis con la que se pudiera desarrollar la funcionalidad necesaria y se adoptó una semántica basada en lógica descriptiva que permitiese procesos de inferencia sobre los datos para extraer el conocimiento necesario en cada caso. De esta manera, y con la ayuda de servicios web como herramientas comunicadoras, CARL ha podido conocer las necesidades de las aplicaciones y darles solución de manera automática, suponiendo un avance en relación a los trabajos previos existentes en el área.

Para que CARL pudiese desarrollar su funcionalidad, ha sido necesario el diseño de un sistema a su alrededor capaz de organizar elementos complementarios a los procesos de comunicación. Para todos ellos se realizó un análisis preliminar de diversas opciones que permitió un proceso de decisión racional para la elección de la mejor alternativa. Tal es el caso del diseño de la arquitectura por capas del sistema, del diseño de la base de datos necesaria para almacenar la información de los usuarios y de la utilización del vocabulario GoodRelations como ontología del dominio, que se ha convertido en una entidad de gran importancia para el funcionamiento de CARL.

Otros módulos complementarios de especial relevancia para proporcionar la interoperabilidad a través de CARL son las figuras del motor y del intérprete. Éste último se encarga, entre otras cosas, de establecer las equivalencias entre CARL y la semántica de OWL-DL que permiten usar razonadores de lógica descriptiva y sin el cual la sintaxis definida carecería de sentido. Por otro lado, el motor gestiona los procesos de comunicación junto con CARL y realiza las conversiones de datos precisas en cada momento para adaptar los diferentes formatos de las aplicaciones. Como se puede comprobar, la aportación de estas entidades es clave y ambas resultan insustituibles para alcanzar los objetivos planteados en la investigación.

El proceso de validación de los distintos aspectos del lenguaje y sistema propuestos ha demostrado las hipótesis planteadas, probando así la viabilidad y eficacia de la solución. Con este proceso se ha confirmado, entre otras cosas, que un lenguaje semántico basado en

ontologías es una posible solución al problema de la interoperabilidad, que los entornos proporcionados por el paradigma del Cloud Computing permiten crear entornos cerrados donde establecer prerrequisitos y que gracias a las tecnologías semánticas se pueden automatizar los procesos de intercambio de información. Con los resultados obtenidos se puede concluir que la solución propuesta ha cumplido los objetivos de la investigación y que con ella se han realizado aportaciones de interés para el sector y la comunidad investigadora, presentando una solución innovadora y eficaz a los problemas objetos de estudio.

El desarrollo de esta tesis doctoral ha supuesto un paso más en las líneas de investigación estudiadas y ha conllevado un trabajo organizado, meticuloso y exhaustivo. Debido en parte a ello, también han surgido una serie de problemas que deben mencionarse y que, una vez resueltos, han contribuido a incrementar la calidad de la investigación. En primer lugar, el desconocimiento inicial del autor en la materia llevó a realizar un estudio muy amplio de tecnologías y enfoques posibles con los que afrontar los objetivos que se planteaban. Este hecho dificultó las primeras fases de un trabajo ya de por sí complicado por las diferentes áreas de investigación que reúne. Por otro lado, la creación de un lenguaje siempre es una tarea difícil debido a la necesaria definición inicial de la funcionalidad y los prerrequisitos a cumplir que deben ser posteriormente contemplados por la sintaxis y la semántica del mismo. Estas acciones elevan la complejidad de la creación de un lenguaje que, además, debe interactuar con otros elementos en un mismo ecosistema y permitir operaciones de inferencia lógica. Por último, otro de los problemas que surgieron fue la configuración de la metodología de validación. Los objetivos de la investigación se centran en torno a la creación de un lenguaje que permita el intercambio de información entre aplicaciones, objetivos que son difíciles de cuantificar ya que no se basan en parámetros numéricos. Por ello se decidió diseñar un sistema que permitiese mostrar que las operaciones llevadas a cabo se realizaban correctamente y que los diferentes entornos, tecnologías y conceptos utilizados contribuían a la consecución de esos objetivos. Como se mencionaba anteriormente, estas dificultades encontradas en el proceso han provocado en último término un salto cualitativo que se refleja en los resultados obtenidos por la investigación y han permitido avanzar en una dirección de gran interés para la comunidad investigadora. Además, las conclusiones que se pueden extraer de esta tesis doctoral pueden motivar la continuación de trabajos en esta línea, realizando investigaciones paralelas en las mismas áreas.

Como conclusión final se puede afirmar que con esta investigación se han conseguido cumplir los objetivos propuestos y la solución planteada representa una aproximación válida para afrontar el problema de la interoperabilidad entre aplicaciones heterogéneas a nivel de datos. La documentación presentada refleja el resultado de una ardua investigación que ha cubierto un amplio estado del arte en las materias de relevancia para el desarrollo del trabajo, la fijación de unos objetivos y unas hipótesis que han guiado el proceso investigador, aspectos de diseño del lenguaje semántico y del sistema en el que se insertará y una validación de las hipótesis que ha probado cada uno de los axiomas expuestos. Por todo ello se puede decir que la tesis doctoral ha concluido con éxito y que se ha realizado una contribución de interés en el ámbito de la investigación realizada.

Finalmente y como valoración personal, querría decir que el desarrollo de esta tesis doctoral ha resultado más enriquecedor de lo que podría imaginar. Si bien es cierto que es un trabajo ciertamente complicado y en ocasiones duro, creo que me ha ayudado a crecer como

profesional y como persona. En todas las facetas de mi vida uno de mis objetivos personales es el de sentirme orgulloso de aquello que hago. Una vez concluido este trabajo puedo decir que lo he cumplido. Después de mucho tiempo dedicado a la elaboración de esta investigación, creo que hemos podido aportar nuestro granito de arena en los campos que han sido objetos de estudio. Sin embargo, no es más que eso. Aún queda mucho por hacer. Ojalá éste sea el comienzo de una larga carrera investigadora que me permita seguir trabajando para arrojar luz sobre los problemas y acercarme a utopías como las de un mundo completamente interoperable.

7.2 Líneas Futuras

La presente investigación presenta una solución al problema de la interoperabilidad fundamentada en la creación de un lenguaje basado en tecnologías semánticas. Si bien es cierto que los resultados obtenidos por esta tesis doctoral han sido satisfactorios, existen determinados aspectos que quedan abiertos en cuanto a soluciones alternativas, posibles mejoras y líneas de investigación relacionadas.

En primer lugar, la solución que aquí se presenta es una de las posibles dentro de un abanico considerable de opciones. A lo largo del trabajo se han justificado las diferentes decisiones tomadas para llegar a la solución propuesta. Estas decisiones se han tomado en base a aspectos objetivos, pero también en base a una serie de conceptos que se han considerado prioritarios por encima de otros, con lo que cambiando estas preferencias, se abren diferentes caminos que explorar. Cada una de ellos resulta en una nueva línea en la que trabajar que puede arrojar resultados similares a los aquí mostrados o por el contrario obtener resultados de relevancia en otras direcciones.

Otro aspecto claro que queda para continuar con la investigación es el diseño del propio lenguaje. Uno de los objetivos deseados era que el lenguaje tuviera la mínima sintaxis necesaria para poder gestionar la comunicación entre aplicaciones y sus procesos asociados. En esta línea sería posible añadir o modificar muchos aspectos al lenguaje, tanto a nivel semántico como sintáctico, que proporcionen una nueva dimensión a la solución. Dentro de este campo, el modelo de representación del conocimiento definido en la ontología para describir los diferentes conceptos y relaciones también abre un amplio espectro de futuras líneas de investigación.

Un aspecto que daría más versatilidad al sistema y que podría atraer un mayor número de aplicaciones y usuarios sería el de aumentar el vocabulario de la ontología del dominio. De esta manera, el número de restricciones a nivel de campos se suavizaría de forma que se permitiese una mayor flexibilidad sin perder el control de la información almacenada en el sistema. Sin embargo, y como se ha comentado a lo largo del documento, esto implicaría un aumento de coste de desarrollo, ya que habría que realizar la implementación de un mayor número de funciones de conversión en las que se tuvieran en cuenta los nuevos campos introducidos. Una de las opciones que también se barajaron en esta línea era el de ofrecer la posibilidad a los usuarios de crear nuevos campos que precisen o de solicitar su creación al administrador del sistema. El problema que surgía era el mismo. Si se deja crear el campo al

propietario de la aplicación se complica enormemente el proceso de creación de funciones de transformación además de un lapso de tiempo en el que sería imposible ofrecer la interoperabilidad del nuevo campo. Ofreciendo el caso de solicitar la creación, el proceso se ralentiza, pero se gana en seguridad ya que el administrador puede añadir ese aspecto directamente al vocabulario, estructurándose desde el primer momento. En cualquier caso, estos aspectos quedarían para su estudio en futuras investigaciones.

Precisamente en cuanto a seguridad existen muchas posibilidades. Para el trabajo realizado se han obviado estos aspectos por considerarse fuera de los objetivos principales del mismo. Sin embargo, en determinadas fases sí se plantearon posibilidades para futuras de ampliaciones en este sentido. La primera de ellas era la opción de convertir ciertos datos sensibles (económicos, médicos, etc.) en confidenciales, de forma que las aplicaciones no pudiesen acceder a ellos directamente en la ontología, si no que siguiesen un camino alternativo que proporcionase una mayor seguridad y privacidad a los usuarios. Esto supone también una ampliación en el propio lenguaje, en el diseño de la ontología y en el funcionamiento del sistema, que deben tener en cuenta los nuevos requerimientos. En este sentido existen varias alternativas que quedan abiertas como continuación del trabajo desarrollado y que suponen interesantes líneas futuras de investigación.

Dadas las conclusiones extraídas de la investigación, queda una gran línea de trabajo abierta para aplicar los principios expuestos en esta tesis doctoral a otros entornos donde sea necesario alcanzar la interoperabilidad, es decir, no tan claramente orientado a aplicaciones y servicios. Bien es cierto que la semántica lleva años dedicándose a añadir metadatos a los contenidos para facilitar estos procesos, pero su utilización en esta investigación puede abrir nuevos horizontes en este aspecto.

Por último, mencionar que probablemente una de las líneas más interesantes sobre las que seguir trabajando sería alcanzar la interoperabilidad entre distintos entornos de computación en nube. Teóricamente y siguiendo los principios expuestos, se presenta como algo más que factible, pero sin duda requiere de una serie de estudios minuciosos para que se pueda esbozar una solución eficaz al problema. A pesar de que existen grandes probabilidades, no es una tarea trivial ya que el trabajo se enfrentaría a los problemas de amplitud de solución que históricamente han hecho fracasar propuestas anteriores en esta línea. Sin embargo, de lograrse, sus posibilidades serían enormes, dado que se obtendrían los mismos beneficios que los que se presentan en esta tesis doctoral pero a una mayor escala, suponiendo un gran avance hacia la consecución de un mundo interoperable.

7.3 Publicaciones Realizadas

Como consecuencia de esta investigación, se han realizado las siguientes publicaciones en diferentes ámbitos científicos:

- **CARL: A Complex Applications Interoperability Language based on Semantic Technologies for Platform-as-a-Service Integration and Cloud Computing**. E. Jiménez-

Domingo, J.M. Gómez-Berbís, R. Colomo-Palacios, Á. García-Crespo. *Journal of Research and Practice in Information Technology, vol. 43 (3) (2012).*

- **Improving Accuracy of Decision Trees Using Clustering Techniques**. Javier Torres Niño, Alejandro Rodríguez González, Ricardo Colomo Palacios, Enrique Jiménez-Domingo, Giner Alor-Hernández. *J. UCS 19(4): 484-501 (2013)*

- **AKNOBAS: A knowledge-based segmentation recommender system based on intelligent data mining techniques**. Alejandro Rodríguez González, Javier Torres Niño, Enrique Jiménez Domingo, Juan Miguel Gómez Berbís, Giner Alor Hernández. *Comput. Sci. Inf. Syst. 9(2): 713-740 (2012)*

- **Using agents to parallelize a medical reasoning system based on ontologies and description logics as an application case**. Alejandro Rodríguez González, Javier Torres Niño, Gandhi Hernández-Chan, Enrique Jiménez-Domingo, José María Álvarez-Rodríguez. *Expert Syst. Appl. 39(18): 13085-13092 (2012)*

- **Using Ontologies in drug prescription: The SemMed approach**. A. Rodríguez-González, Á. García-Crespo, R. Colomo-Palacios, J. M. Gómez-Berbís, E. Jiménez Domingo. *International Journal of Knowledge-Based Organizations, vol. 1 (4), 2011.*

- **Multi-Tenancy: A new architecture for clouds** (Book Chapter). E. Jiménez-Domingo, A. Lagares-Lemos, J. M. Gómez Berbís, in *Cloud computing: methodology, system, and applications, CRC Press (2011).*

- **CLOUDIO. A Cloud Computing-oriented Multi-Tenant Architecture for Business Information Systems**. E. Jiménez Domingo, J. Torres Niño, Á. Lagares Lemos, M. Lagares Lemos, J. M. Gómez Berbís, R. Colomo Palacios. *IEEE Cloud 2010, Miami, USA, July 5-10. 2010.*

Apéndice A

Abreviaturas

Abreviatura	Significado
AICPA	American Institute of Certified Public Accounts
AJAX	Asynchronous JavaScript and XML
AMI	Amazon Machine Image
API	Application Programming Interface
AT	Application Type
ATHENA	Adavanced Technology for Interoperability of Heterogeneous Enterprise Network and their Applications
AWS	Amazon Web Services
B2B	Business to Business
B2C	Business to Consumer
BD	Base de Datos
BBDD	Bases de Datos
BIF	Bussines Information Framework
BORO	Business Object Reference Ontology
BPEL	Business Process Execution Language
BPEL4WS	Business Process Executions Language for Web Services
BPSS	Business Process Specification Schema
CA	Aplicación Compleja
CAP	Composed Application
CARL	Complex Application and Representation Language

CC	Conformidad Conceptual
CD	Compact Disc
CIF	Código de Identificación Fiscal
CPD	Centro de Procesamiento de Datos
CSS	Cascading Style Sheet
DAML	DARPA Agent Markup Language
DC	Dublin Core
DCMI	Dublin Core Metadata Initiative
DLP	Description Logic Programs
DRM	Digital Rights Management
DT	Data Type
E/S	Entrada/Salida
EBIF	eBussines Interoperability Forum
EBNF	Extended Backus-Nour Form
ebXML	Electronic Business using eXtensible Markup Language
EC2	Elastic Compute Cloud
EDA	Event Driven Architecture
EIC	European Interoperability Centre
ESB	Enterprise Service Bus
FF	Fundamentos Formales
FIFO	First In First Out
FOAF	Friend Of A Friend
GUI	Graphical User Interface
GUID	Globally Unique Identifier

HIPAA	Health Insurance Portability and Accountability Act
HPC	High Performance Computing
HTML	HyperText Markup Language
IA	Inteligencia Artificial
IaaS	Infraestructure as a Service
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
INTEROP	Interoperability Research Networks and their Applications
IOPEs	Inputs, Outputs, Preconditions and Effects
IRS	Internet Reasoning Service
ISO	International Organization for Standardization
JDBC	Java Database Connectivity
JMS	Java Message Service
KR	Knowledge Representation
LOPD	Ley Orgánica de Protección de Datos
MDA	Model Driven Architecture
MEP	Manufacturing Enterprise Processes
MVC	Modelo Vista Controlador
M&S	Marntenaince and Services
NAICS	North American Industry Classification System
NIF	Número de Identificación Fiscal
OASIS	Organization for the Advancement of Structure Information Standards
OCCI-WG	Open Cloud Computing Interface Working Group

OIL	Ontology Inference Layer
OVF	Open Virtualization Format
OWL	Web Ontology Language
OWL-DL	Web Ontology Language Description Logics
OWL-S	Semantic Markup for Web Services
PaaS	Platform as a Service
PC	Personal Computer
PCI DSS	Payment Card Industry Data Security Standard
PDA	Personal Digital Assistant
PICS	Platforms for Internet Content Selection
PYMES	Pequeñas y Medianas Empresas
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
RDQL	RDF Data Query Language
RDLOPD	Reglamento de Desarrollo de la Ley Orgánica de Protección de Datos
RDS	Relational Database Source
REST	Representational State Transfer
RFQ	Request for Quotation
S3	Simple Storage Service
SaaS	Software as a Service
SAWSDL	Semantic Annotations for Web Service Description Language
SBB	Service Building Block
SDK	Software Development Kit

SGML	Standard Generalized Markup Language
SGSI	Sistema de Gestión de la Seguridad de la Información
SIC	Sistema de Información Corporativos
SIC	Standard Industry Classification
SMTP	Simple Mail Transfer Protocol
SLA	Service Level Agreement
SO	Sistema Operativo
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	Protocol and RDF Query Language
SQL	Structured Query Language
SQS	Simple Qeue Service
SWRL	Semantic Web Rule Language
SWS	Servicios Web Semánticos
SWSF	Semantic Web Services Framework
TI	Tecnologías de la Información
TIC	Tecnologías de la información y la comunicación
UDDI	Universal Description Discovery and Integration
UE	Unión Europea
UK	United Kingdom
UML	Unified Modeling Language
UNECE	United Nations Economic Comission for Europe
URI	Uniform Resource Identifier
VM	Virtual Machine

W3C	World Wide Web Consortium
WiMAX	Worldwide Interoperability for Microwave Access
WS-CDL	Web Services Choreography Description Language
WS-I	Web Service Interoperability
WSCI	Web Service Choreography Interface
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WSMF	Web Service Modeling Framework
WSMO	Web Service Modeling Ontology
WSMT	Web Service Modeling Toolkit
WSMX	Web Service Modelling eXecution Environment
WWW	World Wide Web
XML	Extensible Markup Language
XSD	XML Schema Definition

Bibliografía

Aalst, W. M. P. van der. (2003). Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*.

Abadi, M., & Loo, B. T. (2007). Towards a declarative language and system for secure networking. In *Proceedings of the 3rd USENIX international workshop on Networking meets databases* (p. 2).

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). Web Services. In *Web Services* (pp. 123–149). Springer Berlin Heidelberg.

Alvaro, P., Condie, T., Conway, N., Elmeleegy, K., Hellerstein, J. M., & Sears, R. (2010). Boom analytics: exploring data-centric, declarative programming for the cloud. In *Proceedings of the 5th European conference on Computer systems* (pp. 223–236).

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Weerawarana, S. (2003). *Business Process Execution Language for Web Services Version 1.1*. BEA, IBM, Microsoft, SAP, Siebel.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Zaharia, M. (2009). *Above the Clouds: A Berkeley View of Cloud Computing* (No. UCB/EECS-2009-28). EECS Department, University of California, Berkeley. Retrieved from http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, *53*(4), 50–58.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, P. Cudré-Mauroux (Eds.), *The Semantic Web* (pp. 722–735). Springer Berlin Heidelberg.

Baader, F. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Benjamins, V. R., Davies, J., Baeza-Yates, R., Mika, P., Zaragoza, H., Greaves, M., … Fensel, D. (2008). Near-Term Prospects for Semantic Technologies. *Intelligent Systems*, *23*(1), 76–88.

Benslimane, D., Dustdar, S., & Sheth, A. (2008). Services Mashups: The New Generation of Web Applications. *IEEE Internet Computing*, *12*(5), 13–15.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*.

Berners-Lee, Tim. (1998). Semantic Web Road map. *Design Issues for the World Wide Web*, *2008* (September 1998), 1–10.

Berners-Lee, Tim, Cailliau, R., Groff, J.-F., & Pollermann, B. (1992). World-Wide Web: The Information Universe. *Internet Research*, *2*(1), 52–58.

Berre, A.-J., Elvesæter, B., Figay, N., Guglielmina, C., Johnsen, S. G., Karlsen, D., … Lippe, S. (2007). The ATHENA Interoperability Framework. In R. J. Gonçalves, J. P. Müller, K. Mertins, & M. Zelm (Eds.), *Enterprise Interoperability II* (pp. 569–580). Springer London.

Berson, A. (1996). *Client/server architecture (2nd ed.)*. New York, NY, USA: McGraw-Hill, Inc.

Bertocco, M., Ferraris, F., Offelli, C., & Parvis, M. (1998). A client-server architecture for distributed measurement systems. *IEEE Transactions on Instrumentation and Measurement*, *47*(5), 1143–1148.

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - The Story So Far: *International Journal on Semantic Web and Information Systems*, *5*(3), 1–22.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009). DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, *7*(3), 154–165.

Booth, D., & Liu, C. K. (2007). *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*.

Bottaro, A., & Gérodolle, A. (2008). Home SOA -: facing protocol heterogeneity in pervasive applications. In *Proceedings of the 5th international conference on Pervasive services* (pp. 73–80). New York, NY, USA: ACM.

Brown, J., Lampson, B., & Liu, W. (1998). On the refinement of Web services. *Journal of Linear-Time, Mobile Epistemologies*, *5*, 76–88.

Bruijn, J. D., Lara, R., Polleres, A., & Fensel, D. (2005). *OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the Semantic Web*.

Burbeck, S. (1987). *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. Retrieved from http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

Burstein, M., Hobbs, J., Lassila, O., Mcdermott, D., Mcilraith, S., Narayanan, S., Sycara, K. (2004, November). OWL-S: Semantic Markup for Web Services. *W3C Member Submission*.

Bussler, C., Cimpian, E., Fensel, D., Gomez, J., Haller, A., Haselwanter, T. (2005). *Web Service Execution Environment (WSMX), W3C Member Submission, June 2005*.

Buyya, R. (2009). Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID '09* (p. 1). Presented at the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID '09.

Buyya, R., Yeo, C. S., & Venugopal, S. (2008). Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *10th IEEE International Conference on High Performance Computing and Communications, 2008. HPCC '08* (pp. 5–13). Presented at the 10th IEEE International Conference on High Performance Computing and Communications, 2008. HPCC '08.

Buyya, Rajkumar, Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, *25*(6), 599–616.

Campbell-Kelly, M. (2009). Historical reflections: The rise, fall, and resurrection of software as a service. *Commun. ACM*, *52*(5), 28–30.

Carvin, A. (2005). Tim Berners-Lee: Weaving a Semantic Web. In *A Sense Of Place Network*.

Casola, V., Rak, M., & Villano, U. (2010). Identity federation in cloud computing. In *2010 Sixth International Conference on Information Assurance and Security (IAS)* (pp. 253–259). Presented at the 2010 Sixth International Conference on Information Assurance and Security (IAS).

Castells, P. (2003). La Web Semántica. *Sistemas Interactivos y Colaborativos en la Web.*

Chandrasekaran, B., Josephson, J. R., & Benjamins, V. R. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems and their Applications*, *14*(1), 20–26.

Chang, W. Y., Abu-Amara, H., & Sanford, J. F. (2010). *Transforming Enterprise Cloud Services*. Springer.

Chen, D., & Daclin, N. (2010). Framework for Enterprise Interoperability. In Hervé Panetto & N. Boudjlida (Eds.), *Interoperability for Enterprise Software and Applications* (pp. 77–88). ISTE. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/9780470612200.ch6/summary

Chen, D., & Doumeingts, G. (2003). European initiatives to develop interoperability of enterprise applications—basic concepts, framework and roadmap. *Annual Reviews in Control*, *27*(2), 153–162.

Chohan, N., Bunch, C., Pang, S., Krintz, C., Mostafa, N., Soman, S., & Wolski, R. (2010). AppScale: Scalable and Open AppEngine Application Development and Deployment. In D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, & E. Dekel (Eds.), *Cloud Computing* (pp. 57–70). Springer Berlin Heidelberg.

Chomsky, N. (1965). *Aspects of the theory of syntax*. Cambridge, Massachusetts: M.I.T. Press.

Colomo-Palacios, Ricardo, García-Crespo, Á., Soto-Acosta, P., Ruano-Mayoral, M., & Jiménez-López, D. (2010). A case analysis of semantic technologies for R&D intermediation information management. *International Journal of Information Management*, *5*(30), 465–469.

Concur. (2012). *http://www.concur.com/*. Retrieved from http://www.concur.com/

Cunningham, P., Cunningham, P. M., & Fatelnig, P. (2003). *Building the Knowledge Economy: Issues, Applications, Case Studies*. IOS Press.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, *6*(2), 86–93.

Czarnecki, K., & Eisenecker, U. W. (2000). Generative Programming (pp. 2–19). Addison-Wesley.

Davies, J., Studer, R., & Warren, P. (Eds.). (2006). *Semantic Web Technologies: Trends and Research in Ontology-Based Systems*. Chichester, UK: Wiley.

Davulcu, H., Freire, J., Kifer, M., & Ramakrishnan, I. V. (1999). A layered architecture for querying dynamic Web content. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data* (pp. 491–502). New York, NY, USA: ACM.

De, J., Holger, B., Polleres, L. A., Fensel, D., De, J., Holger, B., … Fensel, P. D. (2005). *THE WEB SERVICE MODELING LANGUAGE WSML: AN OVERVIEW*.

Dekel, U., Cohen, T., & Porat, S. (2003). Towards a standard family of languages for matching patterns in source code. In *Software: Science, Technology and Engineering, 2003. SwSTE'03. Proceedings. IEEE International Conference on* (pp. 10–19).

Dieng, R., Corby, O., Giboin, A., Ribière, M., & Lucioles, R. D. (1998). *Methods and Tools for Corporate Knowledge Management*.

Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B., & Pedrinaci, C. (2008). IRS-III: A broker-based approach to semantic Web services. *Journal of Web Semantics*, *6*(2), 109–132.

Donaldson, L. (2001). *The Contingency Theory of Organizations*. SAGE.

Donini, F. M., Lenzerini, M., Nardi, D., & Schaerf, A. (1997). *Reasoning in Description Logics*.

Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., & Senanayake, R. (2005). The owl-s editor - a development tool for semantic web services. In *In Proceedings of the Second European Semantic Web Conference* (pp. 78–92). Springer.

Engel, T., Granzer, H., Koch, B. F., Winter, M., Sampatakos, P., Venieris, I. S., … Salsano, S. (2003). AQUILA: adaptive resource control for QoS using an IP-based layered architecture. *IEEE Communications Magazine*, *41*(1), 46–53.

Erl, T. (2005). *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Fensel, D., & Musen, M. A. (2001). The semantic web: a brain for humankind. *Intelligent Systems*, *16*(2), 24–25.

Fensel, Dieter. (2001). Ontologies. In *Ontologies* (pp. 11–18). Springer Berlin Heidelberg.

Fensel, Dieter. (2003). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Heidelberg: Springer-Verlag.

Fensel, Dieter, & Bussler, C. (2002). The Web Service Modeling Framework WSMF (pp. 17–20).

Fensel, Dieter, Lausen, H., Polleres, A., Bruijn, J. de, Stollberg, M., Roman, D., & Domingue, J. (2006). *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Fernandez, M., d' Aquin, M., & Motta, E. (2011). Linking Data across Universities: An Integrated Video Lectures Dataset. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, E. Blomqvist (Eds.), *The Semantic Web – ISWC 2011* (pp. 49–64). Springer Berlin Heidelberg.

Fernández-Breis, J.T. (2003). *Un entorno para Integrar Ontologías para el Desarrollo de Sistemas de Gestión de Conocimiento*.

Fernández-Breis, Jesualdo Tomás, Castellanos-Nieves, D., & Valencia-García, R. (2009). Measuring individual learning performance in group work from a knowledge integration perspective. *Inf. Sci.*, *179*(4), 339–354.

Fisher, S. (2007). The architecture of the apex platform, salesforce. com's platform for building on-demand applications. In *Software Engineering-Companion, 2007. ICSE 2007 Companion. 29th International Conference on* (pp. 3–3).

Fitzgerald, M. N., Foley, S. N., & Foghlu, M. O. (2009). Network Access Control Configuration Management using Semantic Web Techniques. *Journal of Research and Practice in Information Technology*.

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop, 2008. GCE '08* (pp. 1 –10). Presented at the Grid Computing Environments Workshop, 2008. GCE '08.

Fu, X., Bultan, T., & Su, J. (2004). Analysis of interacting BPEL web services. In *Proceedings of the 13th international conference on World Wide Web* (pp. 621–630). New York, NY, USA: ACM.

Galinec, D. (2010). Human Capital Management Process Based on Information Technology Models and Governance. *International Journal of Human Capital and Information Technology Professionals (IJHCITP)*, *1*(1), 44–60.

García-Crespo, Á., Colomo-Palacios, R., Gómez-Berbís, J. M., & Mencke, M. (2009). BMR: Benchmarking Metrics Recommender for Personnel issues in Software Development Projects. *International Journal of Computational Intelligence Systems*, *3*(2), 257–267.

García-Crespo, Á., Colomo-Palacios, R., Gómez-Berbís, J. M., & Ruiz-Mezcua, B. (2010). SEMO: a framework for customer social networks analysis based on semantics. *Journal of Information Technology*, *25*(2), 178–188.

García-Crespo, Á., Gómez-Berbís, J. M., Colomo-Palacios, R., & Alor-Hernández, G. (2011). SecurOntology: A semantic web access control framework. *Comput. Stand. Interfaces*, *33*(1), 42–49.

Garcia-Sanchez, F., Fernandez-Breis, E., Valencia-Garcia, R., Jimenez, E., Gomez, J. M., Torres-Niño, J., & Martinez-Maqueda, D. (2009). SITIO: semantic business processes based on software-as-a-service and cloud computing. In *Proceedings of the 8th WSEAS International Conference on E-Activities and information security and privacy* (pp. 130–135). Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS).

Garcia-Sanchez, F., Fernandez-Breis, E., Valencia-Garcia, R., Jimenez, E., Gomez, J. M., Torres-Niño, J., & Martinez-Maqueda, D. (2010). Adding semantics to software-as-a-service and cloud computing. *W. Trans. on Comp.*, *9*(2), 154–163.

Garciasanchez, F., Fernandezbreis, J., Valenciagarcia, R., Gomez, J., & Martinezbejar, R. (2008). Combining Semantic Web technologies with Multi-Agent Systems for integrated access to biological resources. *Journal of Biomedical Informatics*, *41*(5), 848–859.

Garfinkel, S. L., & Garfinkel, S. L. (2007). *An Evaluation of Amazon's Grid Computing Services: EC2, S3, and SQS*. Center for.

Geraci, A. (1991). *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. (F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, … F. Springsteel, Eds.). Piscataway, NJ, USA: IEEE Press.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, *5*(2), 199–220.

Guédria, W., Chen, D., & Naudet, Y. (2009). A Maturity Model for Enterprise Interoperability. In *Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK 2009* (pp. 216–225). Berlin, Heidelberg: Springer-Verlag.

Guijarro, L. (2007). Interoperability frameworks and enterprise architectures in e-government initiatives in Europe and the United States. *Government Information Quarterly*, *24*(1), 89–101.

Hanus, M. (2007). Multi-paradigm declarative languages. In *Proceedings of the 23rd international conference on Logic programming* (pp. 45–75).

Hay, B., Nance, K., & Bishop, M. (2011). Storm Clouds Rising: Security Challenges for IaaS Cloud Computing. In *2011 44th Hawaii International Conference on System Sciences (HICSS)* (pp. 1–7). Presented at the 2011 44th Hawaii International Conference on System Sciences (HICSS).

Hepp, M. (2008). GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In A. Gangemi & J. Euzenat (Eds.), *Knowledge Engineering: Practice and Patterns* (pp. 329–346). Springer Berlin Heidelberg.

Hess, A., Johnston, E., & Kushmerick, N. (2004). ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services. In *In Intl. Semantic Web Conf. (ISWC*.

Hoare, C. A. R. (1978). Communicating sequential processes. *Commun. ACM*, *21*(8), 666–677.

Horrocks, I. (2002). DAML+OIL: a Description Logic for the Semantic Web. *IEEE Data Engineering Bulletin*, *25*, 4–9.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. World Wide Web Consortium.

Huhns, M. N., & Singh, M. P. (1998). *Readings in Agents*. Morgan Kaufmann.

Intalio, A. A., Intalio, S. A., Fordin, S., Sap, W. J., Kawaguchi, K., Orchard, D., Microsystems, S. (2002). *Web Service Choreography Interface 1.0*.

Jamcracker. (2012). *http://www.jamcracker.com/*. Retrieved from http://www.jamcracker.com/

Janssen, M., & Joha, A. (2011). CHALLENGES FOR ADOPTING CLOUD-BASED SOFTWARE AS A SERVICE (SAAS) IN THE PUBLIC SECTOR. *ECIS 2011 Proceedings*.

Jiang, X., & Tan, A.-H. (2009). Learning and inferencing in user ontology for personalized Semantic Web search. *Inf. Sci.*, *179*(16), 2794–2808.

Ju, J., Wang, Y., Fu, J., Wu, J., & Lin, Z. (2010). Research on Key Technology in SaaS. In *2010 International Conference on Intelligent Computing and Cognitive Informatics (ICICCI)* (pp. 384–387). Presented at the 2010 International Conference on Intelligent Computing and Cognitive Informatics (ICICCI).

Kasunic, M. (2001). *Measuring Systems Interoperability: Challenges and Opportunities*.

Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., & Barreto, C. (2005). *Web Services Choreography Description Language Version 1.0*.

Kearney, K. T., & Torelli, F. (2011). The SLA Model. In P. Wieder, J. M. Butler, W. Theilmann, & R. Yahyapour (Eds.), *Service Level Agreements for Cloud Computing* (pp. 43–67). Springer New York.

Keller, E., & Rexford, J. (2010). The "Platform as a service" model for networking. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking* (pp. 4–4). Berkeley, CA, USA: USENIX Association.

Kerrigan, M., Mocan, A., Tanler, M., & Fensel, D. (2007). The Web Service Modeling Toolkit - An Integrated Development Environment for Semantic Web Services (System Description. In *In Proc. of the 4th European Semantic Web Conference (ESWC* (pp. 789–798).

Khriyenko, O., Terziyan, V., & Kaikova, O. (2013). End-user Facilitated Interoperability in Internet of Things: Visually-enriched User-assisted Ontology Alignment. *International Journal On Advances in Internet Technology*, *6*(1 and 2), 90–100.

Kim, W., & Seo, J. (1991). Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, *24*(12), 12–18.

Klyne, G., & Carroll, J. J. (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*.

Knorr, E., & Gruman, G. (2008). What cloud computing really means. *InfoWorld*, *7*.

Kruchten, P., Obbink, H., & Stafford, J. (2006). The Past, Present, and Future for Software Architecture. *IEEE Software*, *23*(2), 22–30.

Lagoze, C., & Van de Sompel, H. (2001). The open archives initiative: building a low-barrier interoperability framework. In *Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries* (pp. 54–62). New York, NY, USA: ACM.

Lakshmanan, L. V. S., Sadri, F., & Subramanian, I. N. (1996). A declarative language for querying and restructuring the Web. In *Research Issues in Data Engineering, 1996. Interoperability of Nontraditional Database Systems. Proceedings. Sixth International Workshop on* (pp. 12–21).

Lara, R., Roman, D., Polleres, A., & Fensel, D. (2004). A Conceptual Comparison of WSMO and OWL-S. In L.-J. Zhang (Ed.), *ECOWS* (Vol. 3250, pp. 254–269). Springer.

Lausen, H., de Bruijn, J., Polleres, A., & Fensel, D. (2005). WSML - a Language Framework for Semantic Web Services. In *W3C Workshop on Rule Languages for Interoperability*.

Lawton, G. (2008). Developing Software Online With Platform-as-a-Service Technology. *Computer*, *41*(6), 13–15.

Leff, A., & Rayfield, J. T. (2001). Web-application development using the Model/View/Controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International* (pp. 118–127). Presented at the Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International.

Li, K., Verma, K., Mulye, R., Rabbani, R., Miller, J. A., & Sheth, P. (2006). *Designing Semantic Web Processes: The WSDL-S Approach*.

Litwin, W., & Abdellatif, A. (1986). Multidatabase Interoperability. *Computer*, *19*(12), 10–18.

Lu, Y., & Sun, B. (2009). The Fitness Evaluation Model of SAAS for Enterprise Information System. In *IEEE International Conference on e-Business Engineering, 2009. ICEBE '09* (pp. 507–511). Presented at the IEEE International Conference on e-Business Engineering, 2009. ICEBE '09.

Maderuelo Fernández, J. A. (2002). Gestión de la calidad total: El modelo EFQM de excelencia. *Medifam*, *12*(10), 41–54. doi:10.4321/S1131-57682002001000004

MarcoIberoamericano. (2010). Retrieved May 15, 2013, from https://www.google.es/search?q=%22Bases+para+una+estrategia+iberoamericana+de+interoperabilidad%22&aq=f&oq=%22Bases+para+una+estrategia+iberoamericana+de+interoperabilidad%22&aqs=chrome.0.57j60j0j62l3.391j0&sourceid=chrome&ie=UTF-8

Martin, D., Paolucci, M., Mcilraith, S., Burstein, M., Mcdermott, D., Mcguinness, D., … Sycara, K. (2004). Bringing Semantics to Web Services: The OWL-S Approach (pp. 26–42). Springer.

McBride, B. (2001). *Jena: Implementing the RDF Model and Syntax Specification*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.4608

McCarthy, J. (1961). A basis for a mathematical theory of computation, preliminary report. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference* (pp. 225–238). New York, NY, USA: ACM.

McGuinness, D. L., & van Harmelen, F. (2004). *OWL Web Ontology Language Overview*. W3C - World Wide Web Consortium. Retrieved from http://www.w3.org/TR/owl-features/

McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic Web services. *IEEE Intelligent Systems*, *16*(2), 46–53.

Melnik, S., Rahm, E., & Sosna, D. (2001). DOL: An Interoperable Document Server. In *GI Jahrestagung (1)* (pp. 368–377). Retrieved from http://dblp.uni-trier.de/db/conf/gi/gi2001-1.html#MelnikRS01

Miles, A., Matthews, B., Wilson, M., & Brickley, D. (2005). SKOS Core: Simple knowledge organisation for the Web. *International Conference on Dublin Core and Metadata Applications*, *0*(0), pp. 3–10.

Miller, L., Seaborne, A., & Reggiori, A. (2002). Three Implementations of SquishQL, a Simple RDF Query Language. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web* (pp. 423–435). London, UK: Springer-Verlag.

Morris, E., Levine, L., Meyers, C., Place, P., & Plakosh, D. (2004). *System of Systems Interoperability (SOSI): Final Report*.

Navas-Delgado, I., Kerzazi, A., & Aldana-Montes. (2004). *Un Editor de Modelos OWL-S: OWL-S Modeller*.

Netsuite. (2012). *http://www.netsuite.com/portal/home.shtml*. Retrieved from http://www.netsuite.com/portal/home.shtml

Noy, N. F., & Mcguinness, D. L. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology*.

Nyre, Asmund Ahlmann, & Jaatun, M. G. (2009). Privacy in a Semantic Cloud: What's Trust Got to Do with It? In *Proceedings of the 1st International Conference on Cloud Computing* (pp. 107–118). Berlin, Heidelberg: Springer-Verlag.

Ograph, & Morgens. (2008). Cloud computing. *Communications of the ACM*, *51*(7).

Panetto, H., Berio, G., Benali, K., Boudjlida, N., & Petit, M. (2004). A unified enterprise modelling language for enhanced interoperability of enterprise models.

Panetto, Hervé, & Molina, A. (2008). Enterprise integration and interoperability in manufacturing systems: Trends and issues. *Computers in Industry*, *59*(7), 641–646.

Panetto, Hervé, Scannapieco, M., & Zelm, M. (2004). INTEROP NoE: Interoperability Research for Networked Enterprises Applications and Software. In R. Meersman, Z. Tari, & A. Corsaro (Eds.), *On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops* (pp. 866–882). Springer Berlin Heidelberg.

Paolucci, M., Kawamura, T., Payne, T. R., & Sycara, K. (2002). Importing the Semantic Web in UDDI. In Christoph Bussler, R. Hull, S. McIlraith, M. E. Orlowska, B. Pernici, & J. Yang (Eds.), *Web Services, E-Business, and the Semantic Web* (pp. 225–236). Springer Berlin Heidelberg.

Park, J., & Ram, S. (2004). Information systems interoperability: What lies beneath? *ACM Trans. Inf. Syst.*, *22*(4), 595–632.

Partridge, C., & Stefanova, M. (2001). A Synthesis of State of the Art Enterprise Ontologies. In *Lessons Learned. 2001, The BORO Program, LADSEB CNR*.

Pastor, O., Giachetti, G., Marín, B., & Valverde, F. (2013). Automating the Interoperability of Conceptual Models in Specific Development Domains. In I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, & J. Bettin (Eds.), *Domain Engineering* (pp. 349–373). Springer Berlin Heidelberg.

Paulk, M. (2002). Capability Maturity Model for Software. In *Encyclopedia of Software Engineering*. John Wiley & Sons, Inc.

Pawlak, Z. (1981). Information systems theoretical foundations. *Information Systems*, *6*(3), 205–218.

Pedersen, T., Patwardhan, S., & Michelizzi, J. (2004). WordNet::Similarity: measuring the relatedness of concepts. In *Demonstration Papers at HLT-NAACL 2004* (pp. 38–41). Stroudsburg, PA, USA: Association for Computational Linguistics.

Peltz, C. (2003). Web Services Orchestration and Choreography. *IEEE Computer*, *36*(10), 46–52.

Pesic, M., & van der Aalst, W. (2006). A declarative approach for flexible business processes management. In *Business Process Management Workshops* (pp. 169–180).

Powell, A., Nilsson, M., Naeve, A., & Johnston, P. (2005). *DCMI Abstract Model*. Retrieved from http://dublincore.org/documents/abstract-model/

Prud'hommeaux, E., & Seaborne, A. (2008). *SPARQL Query Language for RDF*. World Wide Web Consortium.

Puckett, S. M., & Hensher, D. A. (2009). Revealing the extent of process heterogeneity in choice analysis: An empirical assessment. *Transportation Research Part A: Policy and Practice*, *43*(2), 117 – 126.

REENSKAUG, T. (2003). The Model-View-Controller (MVC) Its Past and Present. *University of Oslo Draft*. Retrieved from http://ci.nii.ac.jp/naid/10019460482/

Rimal, B. P., Choi, E., & Lumb, I. (2009). A Taxonomy and Survey of Cloud Computing Systems. In *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09* (pp. 44–51). Presented at the Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09.

Roman, D., Keller, U., Lausen, H., Bruijn, J. de, Lara, R., Stollberg, M., … Fensel. (2005). Web Service Modeling Ontology. *Applied Ontology*, *1*(1), 77–106.

Ruggaber, R. (2006). ATHENA - Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Applications. In D. Konstantas, J.-P. Bourrières, M. Léonard, & N. Boudjlida (Eds.), *Interoperability of Enterprise Software and Applications* (pp. 459–460). Springer London.

Salesforce. (2011). *http://www.salesforce.com*. Retrieved from http://www.salesforce.com

Sanin, C., Szczerbicki, E., & Toro, C. (n.d.). An OWL Ontology of Set of Experience Knowledge Structure. Retrieved from http://130.203.133.150/viewdoc/similar;jsessionid=718CEA0216F8C6705901A20C32DDA6E4?

Schulz, S., Romacker, M., Faggioli, G., Freiburg, U., & Hahn, U. (1999). From Knowledge Import to Knowledge Finishing: Automatic Acquisition and Semi-Automatic Refinement of Medical Knowledge. In *In Proc. of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop* (p. 99).

Schuster, H., Jablonski, S., Kirsche, T., & Bussler, C. (1994). A client/server architecture for distributed workflow management systems. In *, Proceedings of the Third International Conference on Parallel and Distributed Information Systems, 1994* (pp. 253–256). Presented at the Proceedings of the Third International Conference on Parallel and Distributed Information Systems, 1994.

Schwartz, D. (1999). *When Email Meets Organizational Memories: Addressing Threats to Communication in a Learning Organization*.

Sciore, E., Siegel, M., & Rosenthal, A. (1994). Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Trans. Database Syst.*, *19*(2), 254–290.

Scott, D., & Strachey, C. (1971). Toward A Mathematical Semantics for Computer Languages. In J. Fox (Ed.), (Vol. XXI, pp. 19–46). Presented at the Proceedings of the Symposium on Computers and Automata, Polytechnic Press.

Service Level Agreements for Cloud Computing. (n.d.). Retrieved from http://www.springer.com/computer/communication+networks/book/978-1-4614-1613-5

Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web Revisited. *IEEE Intelligent Systems*, *21*(3), 96–101.

Shahar, Y., & Musen, M. A. (1996). Knowledge-Based Temporal Abstraction in Clinical Domains. *Artifical Intelligence in Medicine,*Vol. 8 (3), Elsevier.

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall.

Sheth, A. P. (1999). Changing Focus on Interoperability in Information Systems:From System, Syntax, Structure to Semantics. In M. Goodchild, M. Egenhofer, R. Fegeas, & C. Kottman (Eds.), *Interoperating Geographic Information Systems* (pp. 5–29). Springer US.

Sheth, A., & Ramakrishnan, C. (2003). Semantic (Web) Technology in Action: Ontology Driven Information Systems for Search, Integration and Analysis. *IEEE Data Engineering Bulletin*, *26*, 40–48.

Simmons, R., Goodwin, R., Haigh, K. Z., Koenig, S., & O'Sullivan, J. (1997). A layered architecture for office delivery robots. In *Proceedings of the first international conference on Autonomous agents* (pp. 245–252). New York, NY, USA: ACM.

Singh, R., Iyer, L. S., & Salam, A. F. (2005). Semantic eBusiness. *Int. J. Semantic Web Inf. Syst.*, *1*(1), 19–35.

Sr, & Sr. (2005). *Event-Driven Architecture Overview - Event-Driven SOA Is Just Part of the EDA Story*. Patricia Seybold Group.

Srinivasan, N., Paolucci, M., & Sycara, K. (2005). *CODE: A Development Environment for OWL-S Web services* (No. CMU-RI-TR-05-48). Pittsburgh, PA: Robotics Institute.

Stevens, R., Goble, C. A., & Bechhofer, S. (2000). *Ontology-based Knowledge Representation for Bioinformatics*.

Strang, T., Linnhoff-Popien, C., & Frank, K. (2003). CoOL: A Context Ontology Language to Enable Contextual Interoperability. In J.-B. Stefani, I. Demeure, & D. Hagimont (Eds.), *Distributed Applications and Interoperable Systems* (Vol. 2893, pp. 236–247). Springer Berlin Heidelberg.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, *25*(1-2), 161–197.

Suchanek, F. M., Kasneci, G., & Weikum, G. (2007). Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web* (pp. 697–706). New York, NY, USA: ACM.

Sure, Y., Prof, R., & Studer, D. R. (2003). *Methodology, Tools & Case Studies for Ontology based Knowledge Management*.

Tanenbaum, A. S., & Steen, M. van. (2006). *Distributed Systems: Principles and Paradigms (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Vaquero, L.M. (2011). EduCloud: PaaS versus IaaS Cloud Usage for an Advanced Computer Science Course. *IEEE Transactions on Education*, *54*(4), 590–598.

Vaquero, L.M., Rodero-Merino, L., Caceres, J., & Lindner, M. (2008). A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, *39*(1), 50–55.

Vaquero, Luis M., Rodero-Merino, L., & Morán, D. (2011). Locking the sky: a survey on IaaS cloud security. *Computing*, *91*(1), 93–118.

Verma, K., & Sheth, A. (2007). Semantically Annotating a Web Service. *IEEE Internet Computing*, *11*, 83–85.

Vorobiev, A., & Bekmamedova, N. (2010). An Ontology-Driven Approach Applied to Information Security. *Journal of Research and Practice in Information Technology*, *42*(1).

Vossen, G., Lytras, M., & Koudas, N. (2007). Editorial: Revisiting the (Machine) Semantic Web: The Missing Layers for the Human Semantic Web. *IEEE Trans. on Knowl. and Data Eng.*, *19*(2), 145–148.

Vouk, M. A. (2008). Cloud computing #x2014; Issues, research and implementations. In *30th International Conference on Information Technology Interfaces, 2008. ITI 2008* (pp. 31–40). Presented at the 30th International Conference on Information Technology Interfaces, 2008. ITI 2008.

Web Services Description Language (WSDL) 1.1, W3C Note. (2001). World Wide Web Consortium (W3C).

Youseff, L., Butrico, M., & Da Silva, D. (2008). Toward a Unified Ontology of Cloud Computing. In *Grid Computing Environments Workshop, 2008. GCE '08* (pp. 1–10).

Zaremba, M., & Vitvar, T. (2008). WSMX: A Solution for B2B Mediation and Discovery Scenarios. In *ESWC* (pp. 884–889).

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, *1*(1), 7–18.

Zhao, G., Deng, J., & Shen, W. (2001). CLOVER: an agent-based approach to systems interoperability in cooperative design systems. *Computers in Industry*, *45*(3), 261–276.

Zheng, H.-T., Kang, B.-Y., & Kim, H.-G. (2008). An ontology-based approach to learnable focused crawling. *Inf. Sci.*, *178*(23), 4512–4522.

Zhou, F., Fang, Y., & Chen, H. (2012). Personalized Travel Service Discovery and Usage in Cloud Environment. In *2012 IEEE Ninth International Conference on e-Business Engineering (ICEBE)* (pp. 333–337). Presented at the 2012 IEEE Ninth International Conference on e-Business Engineering (ICEBE).