Universidad Politécnica de Valencia

I3M

Cloud Computing PaaS Platform Cloud ComPaaS

Tésis de Máster

Valencia, 17 de Septiembre de 2010

Autor:

Andrés García García

Directores:

Dr. D. Vicente Hernández García

D. Carlos de Alfonso Laguna

Índice

Capítulo 1

Introducción

En la literatura aparecen numerosas definiciones de Cloud Computing, realizadas a lo largo del tiempo por distintas empresas y entidades, y que han ido evolucionando en función de sus intereses y al desarrollo del propio concepto. Tras realizar una revisión bibliográfica, nos hemos quedado con una definición de Cloud Computing que resulta bastante exhaustiva y analítica, realizada por Vaquero et al [1], y que es el resultado de las aportaciones de otras definiciones realizadas anteriormente tanto en el ámbito empresarial como científico:

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

Los autores señalan que el mínimo conjunto de características comunes que se asocian al Cloud es virtualización, escalabilidad y modelo de pago por uso, tal y como se señalan en la definición, junto con el concepto de SLA (Service Level Agreement) como elemento contractual entre las partes implicadas.

Adicionalmente, Vaquero et al., en el mismo artículo diferencian tres niveles de Cloud Computing según el recurso ofrecido, desde el nivel de abstracción más bajo al más alto. Estas variantes, conocidas como la pila del Cloud Computing, son Infrastructure-as-a-Service (IaaS), donde el recurso ofrecido son máquinas virtuales, red o almacenamiento (por ejemplo Amazon EC2 [2]), Platform-as-a-Service (PaaS), donde el recurso ofrecido es un entorno de desarrollo y ejecución de aplicaciones y servicios (por ejemplo Google App

Engine [3]) y Software-as-a-Service, donde el recurso ofrecido es una aplicación completa (por ejemplo Force.com [4]). Estas soluciones proveen un nivel de abstracción creciente, desde el nivel hardware hasta aplicaciones software listas para usar a través de Internet.

En los últimos años las tecnologías Cloud Computing y sus aplicaciones han despertado gran interés en los ambientes empresariales. El gran auge y aplicabilidad de esta tecnología ha llevado a la aparición de una variedad de productos comerciales, principalmente orientados al almacenamiento (Amazon S3 [5], Nirvanix [6], Savvis [7]) y a la virtualización de infraestructuras (Amazon EC2, GoGrid [8], 3Tera [9]). Siguiendo las tendencias de las empresas, las instituciones académicas han centrado su atención en esta tecnología, generando herramientas de código abierto (Eucalyptus [10], Nimbus [11], OpenNebula [12]) y proyectos de investigación tales como RESERVOIR [13], orientados a solucionar algunos de los problemas inherentes al Cloud Computing. Sin embargo, en el caso de las soluciones comerciales, la mayoría de herramientas y proyectos se han orientado al nivel más bajo de la pila del Cloud Computing (IaaS), lo que ha provocado una ausencia de investigación en los otros dos niveles..Esta situación ha producido una brecha entre el desarrollo del IaaS y los niveles PaaS y SaaS. Por ejemplo, mientras que existen varios proyectos de investigación orientados a resolver problemas asociados con la administración de máquinas virtuales, se ha realizado poco trabajo orientado a la construcción de servicios virtuales desde una perspectiva Cloud.

El Cloud IaaS nace en el seno de las grandes corporaciones del área de las Tecnologías de la Información (IT por sus siglas en inglés) las cuales detectan que un porcentaje importante de los recursos computacionales no están siendo utilizados eficientemente. Estas empresas cuentan con grandes centros de datos con cientos o miles de máquinas, de las que están sacando solamente un pequeño rendimiento. Utilizando el paradigma del Cloud Computing, estas compañías se han convertido en proveedores IaaS obteniendo a cambio los beneficios de la economía de escala, principio por el cual el coste individual de la administración de un recurso disminuye según aumenta el número de estos.

Para construir una nube IaaS, los proveedores utilizan tecnologías de virtualización para particionar su hardware en porciones de tamaños predeterminados en cuanto a cantidad de CPU (número de cores), memoria (gigabytes de RAM) y disco duro (gigabytes de espacio). Mediante la utilización de unas herramientas de administración, los usuarios pueden solicitar imágenes de máquinas virtuales a la plataforma, que se encarga de desplegar la imagen solicitada en el hardware físico disponible y entregar al usuario

la dirección y credenciales de acceso a la misma. Una vez el usuario tiene acceso a la imagen, es libre de utilizarla, pararla o eliminarla por completo, pagando solamente por el tamaño de la instancia solicitada y la cantidad de tiempo que esta ha estado en ejecución. Normalmente se permite acceso total con derechos de administrador a dichas máquinas virtuales.

El público objetivo de las soluciones cloud son pequeñas o medianas empresas. Las empresas pequeñas o de nueva creación (startups) generalmente no poseen centros de datos propios, pues el montaje, administración y mantenimiento de estos necesita de una gran inversión inicial en infraestructura y personal. Utilizando una nube, estas empresas pueden obtener hardware bajo demanda de forma casi instantánea y pagar solamente por la cantidad utilizada, sin preocuparse del mantenimiento de las máquinas. Por otro lado, empresas medianas que ya poseen centros de datos de tamaños reducidos también pueden obtener beneficios de las nubes, al reducir costes migrando sus soluciones actuales a la nube, expandir su negocio sin la necesidad de nuevas inversiones o bien complementar sus soluciones actuales con la nube para superar picos de demanda o un volumen de clientes creciente.

Utilizando este modelo todos los participantes se benefician: los proveedores obtienen rendimiento de unos activos que de otra manera no se hubiesen aprovechado, y los consumidores obtienen acceso a hardware bajo demanda a un coste menor que el de poner en marcha su propio centro de datos, debido al uso de la economía de escala.

Si bien la provisión de máquinas virtuales bajo demanda es un campo bastante trabajado, en el caso de los otros niveles de Cloud Computing no se ha desarrollado lo suficiente. Así, la aproximación de los proveedores Cloud a la virtualización de servicios se basa en muchos casos en el IaaS, y consiste básicamente en ofrecer servicios pre configurados, empaquetados en una imagen de máquina virtual que posteriormente es desplegada en una infraestructura virtual, en lugar de poner en funcionamiento únicamente el servicio requerido. Si bien esta aproximación es sencilla y resulta casi inmediata de aplicar, también conlleva ciertas desventajas como son desaprovechamiento de recursos, administración menos directa de los servicios, bajo nivel de abstracción, etc.

El Cloud PaaS se centra en la provisión de recursos computacionales como una plataforma de ejecución de software, en lugar de máquinas virtuales como el IaaS. Este nivel del Cloud sigue una tendencia natural en el desarrollo de la informática, como es el aumento del nivel de abstracción en los desarrollos. Utilizando una nube PaaS se pasa de tener un sistema operativo básico a tener un entorno de ejecución que proporciona la funcionalidad necesaria para el desarrollo de servicios de alto nivel a través de APIs. En la actualidad

la mayoría de las soluciones PaaS desarrolladas son propietarias y cada una de ellas utiliza distintas arquitecturas, lenguajes, soporte a estándares, etc.

Nuestra aproximación al cloud PaaS consiste en ofrecer una plataforma en la cual sea posible desplegar y administrar servicios virtuales tal y como las plataformas IaaS despliegan y administran máquinas virtuales. Esta aproximación consiste en trasladar los conceptos que se han aplicado de forma eficiente para las nubes IaaS al nivel PaaS, realizando una analogía entre los servicios y las imágenes de máquina virtual, y entre los contenedores de servicios y los *hipervisores*.

Una plataforma PaaS basada en estos principios no se limita a ofrecer los servicios como una capa encima de una nube IaaS, sino que se articula en su conjunto en torno al concepto de servicio y se orienta a ofrecer una serie de características aplicadas a los mismos, como son un sistema de descubrimiento de servicios, un modelo de seguridad, un marco para la definición de SLAs de usuario, mediciones dinámicas, reglas de Calidad de Servicio (QoS), etc.

1.1. Objetivos

A partir de la situación descrita en el apartado anterior, nos planteamos como objetivo general de la Tesis de Máster la especificación de una arquitectura PaaS, y el diseño de una plataforma que, conforme a dicha arquitectura, permita ejecutar y administrar servicios tal y como las plataformas IaaS despliegan y administran máquinas virtuales. Se incluye como objetivo adicional la implementación de un prototipo de dicha plataforma, como prueba de concepto y banco de pruebas de las líneas de investigación planteadas en la Tesis de Máster.

Dentro del marco general de la Tesis de Máster, nos hemos planteado unos objetivos parciales que han guiado el proyecto a través de las distintas fases por las que ha transcurrido. Estos objetivos parciales son la recopilación de las características necesarias para una plataforma PaaS, la identificación de problemas abiertos en el campo de las nubes PaaS y la definición de líneas de investigación a continuar tras la finalización de la Tesis de Máster.

1.2. Desarrollo del proyecto y estructura de la memoria

Para alcanzar los objetivos fijados para la Tesis de Máster, el proyecto ha transitado por una serie de fases que se enumeran a continuación.

- Revisión del estado del arte. En esta fase se realizó un estudio sobre el material informativo, divulgativo y comercial disponible sobre Cloud Computing y tecnologías asociadas. El resultado de esta fase fue la recopilación del conocimiento sobre los últimos desarrollos en tecnologías Cloud, la detección de las debilidades y la identificación de carencias de éstos.

- Especificación de la plataforma PaaS y definición de la arquitectura. En esta fase se partió de las debilidades y carencias identificadas en la revisión del estado del arte para especificar una plataforma PaaS con capacidad para cubrir las mismas. Con el conocimiento existente sobre tecnologías Grid y el nuevo conocimiento adquirido del estudio de soluciones Cloud disponibles, esta fase tuvo como resultado la definición de una arquitectura que representa la especificación de la plataforma.

- Diseño de módulos y componentes. En esta fase se diseñaron los módulos (componentes lógicos que realizan una funcionalidad particular del sistema) y componentes (piezas de software que implementa uno o más módulos) que conforman la arquitectura especificada en la fase anterior, así como sus patrones de interacción. El resultado de esta fase fue el diseño de los elementos que conforman la plataforma.

- Desarrollo de un prototipo del sistema. En esta fase se procedió a desarrollar una implementación acotada del sistema, que sirve como prueba de concepto de la especificación y diseño realizados en anteriores fases. Para ello, se tomó una versión simplificada de la arquitectura y sus módulos y se procedió al desarrollo de los componentes que conforman la misma. El prototipo implementa un subconjunto de la arquitectura puesto que la especificación completa de la plataforma es muy amplia e incluye elementos avanzados tales como los SLA digitales, el marco de seguridad, la interoperabilidad y la auto recuperación, los cuales representan líneas de investigación que sobrepasan el ámbito de la Tesis de Máster y cuyo desarrollo incurre en un coste elevado. El objetivo del prototipo es la obtención de un producto que sirva para comprobar el funcionamiento de los conceptos básicos del diseño de la plataforma y su arquitectura, con un coste temporal reducido. El resultado fue un prototipo funcional de una herramienta para el despliegue de nubes PaaS privadas.

- Elaboración de la memoria de la Tesis de Máster. En esta fase se procedió a elaborar la memoria donde se recopila todo el proceso de ela-

boración de la Tesis de Máster.

Capítulo 2

Perspectiva histórica del Cloud Computing y estado del arte

2.1. Perspectiva histórica

El término Cloud Computing es relativamente reciente, y ha experimentado una evolución de acuerdo a cómo se ha desarrollado la tecnología. El término surge en el año 2001, cuando el periodista John Markoff publicó un artículo en el New York Times [14] donde hizo referencia a la aproximación usada por Microsoft con su plataforma .NET como 'Cloud Computing': el poder acceder a servicios, con independencia del dispositivo utilizado.

Sin embargo, la primera vez que se utilizó Cloud Computing con un significado más similar al que se considera hoy en día, fue en Agosto de 2006, en una conferencia de Google [15]. En ella, Eric Schmidt dijo:

"[...] there is an emergent new model [...] It starts with the premise that the data services and architecture should be on servers. We call it cloud computing - they should be in a "cloud" somewhere. [...] it doesn't matter whether you have a PC or a Mac or a mobile phone [...] you can get access to the cloud. [...] the computation and the data and so forth are in the servers."

Esta definición sienta las piezas básicas que conforman la idea de Cloud Computing: el trabajo se hace en los servidores y los clientes, con independencia de la plataforma, se conectan a una 'nube' abstracta. En esa misma conferencia, citan como proveedores de Cloud Computing a empresas como Amazon, IBM, Microsoft y ellos mismos (Google).

En este contexto se produjo una confusión inherente al hecho de que se trataba de un nuevo paradigma de computación. Así, en Agosto de 2007 John

Markoff escribe un artículo sobre el Cloud Computing [16] donde plantea que la intención de esta tecnología es la siguiente:

"[...] moving computing and data away from the desktop [...] simply displaying the results of computing that takes place in a centralized location and is then transmitted via the Internet"

De nuevo en este texto implica a clientes, servidores y la necesidad de una conexión a Internet. Sin embargo, bajo esta definición tan amplia, otros muchos autores (por ejemplo Conferencia sobre Cloud Computing de la Universidad de Princeton [17], Introducción) tratan de incluir dentro del Cloud Computing elementos como programas P2P, grandes portales web que hacen uso de 'datacenters' (por ejemplo Ebay [18]), juegos online, servicios de correo electrónico, etc.

En base a este criterio nos podemos plantear ¿en qué se diferencia Cloud Computing del modelo cliente-servidor tradicional?, o si un programa de mensajería instantánea se podría considerar como una aplicación Cloud Computing. Ciertamente una definición tan vaga no nos ayuda a ver Cloud Computing como un término que defina algo concreto y novedoso. En este contexto Amazon puso en marcha sus servicios Simple Storage Service (S3) y Elastic Compute Cloud (EC2). Estos dos servicios ofertan, respectivamente, almacenamiento y cómputo en un conjunto de servidores de Amazon, ofreciéndolos como un servicio que se paga mediante una cuota (S3) o de acuerdo a su utilización (EC2). A estos dos elementos se unen otras aplicaciones como Google Docs [19] o Google App Engine, que se convierten en los principales exponentes del Cloud Computing (como vemos la propia Amazon incluye el término en el nombre de sus servicios).

En definitiva, lo que se consigue es dar un nuevo paso hacia el refinamiento de la definición de Cloud Computing, considerando que los elementos que componen la nube son servicios, y que supone una evolución del modelo de comunicación cliente-servidor.

Un poco después, en un informe de David Chappell orientado a empresas [20], acerca del concepto de Cloud Computing, el autor indica:

"Cloud platforms. As its name suggests, this kind of platform lets developers write applications that run in the cloud, or use services provided from the cloud, or both."

De acuerdo a esta nueva definición, la nube ya no es un elemento con el cual conectar cliente y servidor, sino un entorno en el cual se alojan servicios que pueden ser utilizados bajo demanda. Así se plantea desarrollar aplicaciones *en* y *para* la nube. Más adelante, Chappell explica un poco más en profundidad los tres modelos que expone: plataformas Cloud, funcionalidades añadidas y Software as a Service (SaaS).

- Plataformas Cloud: Una plataforma Cloud es un servicio que permite generar aplicaciones en y para la nube. De forma remota nos conectamos a un entorno de desarrollo, en la que podemos crear aplicaciones que a su vez puedan ser accedidas por otros. La plataforma en sí es un elemento Cloud.
- Funcionalidades añadidas: Una aplicación tradicional (es decir, Offline, que se ejecuta en nuestro propio computador) accede a servicios alojados en la nube, que le otorgan alguna funcionalidad nueva.
- Software as a Service (SaaS): Accedemos a ciertas aplicaciones haciendo uso de la nube, por ejemplo, aquellas que nosotros mismos, otros usuarios o los proveedores de la nube han creado.

Este último término es el que aporta un matiz adicional a la definición del Cloud Computing. El hecho es que SaaS no es un concepto nuevo, y hace referencia a una forma de distribuir el software, en la cual en lugar de ofrecerlo como producto, se ofrece como servicio. Un producto software debe ser comprado, instalado por el usuario y finalmente utilizado. En la aproximación del software como servicio, las aplicaciones se alojan en un servidor que es accedido a través de Internet y utilizado bajo demanda del usuario. SaaS forma parte de un paradigma conocido como "Utility Computing", que plantea el consumo de recursos informáticos como servicios, al estilo de la red eléctrica. Si deseamos consumir un recurso (por ejemplo CPU), seguimos los siguientes pasos:

1. Nos conectamos a la 'red de cómputo'.
2. Utilizamos el recurso CPU sin importarnos dónde está alojado o qué elemento está proveyendo esa capacidad de cálculo.
3. Finalmente desconectamos y pagamos por el recurso utilizado.

Llegados a este punto, la confusión sobre la idea de Cloud Computing es notable ya que los pioneros en soluciones Cloud (principalmente Amazon) comienzan a mover un volumen de negocio relevante, y gran número de compañías y grupos de investigación desean sumarse al término de moda y obtener rendimiento de él. De esta forma, algunas compañías comienzan a publicitar sus productos como "Cloud Computing", realizando una adaptación del término a sus propios intereses y deformando el trasfondo real de la tecnología emergente. Este hecho queda patente en las declaraciones de Larry Ellison, CEO de Oracle, que declaraba en el Wall Street Journal [21]:

"The interesting thing about Cloud Computing is that we've redefined Cloud Computing to include everything we already do..."

Para contextualizar la situación del Cloud Computing en aquel momento, el informe Hype Cycle 2009 [22] de Gartner Inc. colocaba a esta tecnología en la cima del ciclo del Hype y le otorgaba un periodo de maduración de 2 a 5 años. A juicio de los analistas, el Cloud Computing se encontraba en el pico de expectativas y popularidad, pero sería necesario un periodo de 2 a 5 años para qué la tecnología llegase al "*Plateau of Productivity*", es decir, el punto en el cual las expectativas exageradas han desaparecido, la tecnología ha madurado y está resultando productiva para las empresas.

Para tratar de poner las cosas en su sitio, desde un punto de vista un poco más formal y externo a los intereses comerciales, la Universidad de Berkeley publicó en Febrero de 2009 el artículo *"Above the Clouds: A Berkeley View of Cloud Computing"* [23]. En este artículo, que podría considerarse seminal desde un punto de vista científico, con respecto a esta tecnología, los investigadores exponen los beneficios del uso del Cloud tanto para las empresas consumidoras (provisión bajo demanda, ausencia de inversión inicial, modelo de pago por uso) como para las proveedoras (obtener beneficio económico, aprovechar la inversión en hardware realizada, establecer un nicho de mercado). Además, se definen conceptos como elasticidad, enunciada como la capacidad de un sistema para escalar de forma automática y con celeridad frente a cambios en la carga de los servicios, y se discuten puntos como los riesgos y oportunidades del Cloud, la conveniencia de la migración a la nube y previsiones sobre el futuro de la tecnología. De esta forma, los investigadores del grupo *"Reliable Adaptative Distributed Systems Laboratory"* trataban de dar un golpe de autoridad aclarando ciertos términos sobre Cloud Computing.

Un aspecto importante es el hecho de que en dicho artículo, los investigadores de Berkeley definen Cloud Computing como:

"Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a Cloud."

En esta definición nos encontramos con cómo los desarrolladores pueden convertirse en proveedores SaaS utilizando soluciones Cloud, estableciendo así, de forma esquemática, la división del Cloud Computing en tres niveles.

En un ambiente donde por fin se empiezan a asentar los conceptos borrosos y poco claros es donde surgen y crecen con fuerza los grandes proveedores de Cloud Computing, bien como nuevas ramas o productos de empresas ya

existentes (Salesforce.com, GoGrid), o bien como empresas nuevas (Engine Yard [24], RightScale [25]). Adicionalmente todas las grandes empresas de informática empiezan a poner en marcha sus propias iniciativas Cloud, como Windows Azure [26] (Microsoft), Google App Engine (Google), Network.com [27] (Sun Microsystem), Blue Cloud [28] (IBM), etc..

En esas mismas fechas, concretamente en Enero de 2009, Vaquero et al. publican el artículo *"A Break in the Clouds: Towards a Cloud Definition"* donde pretenden llegar a una definición consensuada del Cloud Computing y de todos sus términos, dedicándose a recopilar definiciones, extraer elementos comunes y obtener conclusiones. En este artículo se realiza la definición del Cloud que hemos utilizado al principio de la Tesis de Máster, pero un aspecto más importante es que explicita la división de la tecnología en tres niveles, proponiendo la siguiente clasificación.

"IPs manage a large set of computing resources, such as storing and processing capacity. Through virtualization, they are able to split, assign and dynamically resize these resources to build ad-hoc systems as demanded by customers, the SPs. They deploy the software stacks that run their services. This is the Infrastructure as a Service (IaaS) scenario.

Cloud systems can offer an additional abstraction level: instead of supplying a virtualized infrastructure, they can provide the software platform where systems run on. The sizing of the hardware resources demanded by the execu tion of the services is made in a transparent manner. This is denoted as Platform as a Service (PaaS).

Finally, there are services of potential interest to a wide variety of users hosted in Cloud systems. This is an alternative to locally run applications. An example of this is the online alternatives of typical office applications such as word processors. This scenario is called Software as a Service (SaaS)."

A continuación los autores recopilan un total de 22 definiciones realizadas para el Cloud Computing a lo largo del año 2008 y extraen de ellas las características comunes atribuidas al mismo, como son facilidad de uso, virtualización, uso a través de internet, variedad de recursos, adaptación automática, escalabilidad, optimización de recursos, pago por uso y Service Level Agreement (SLA). El resto del artículo presenta una comparación entre el Grid Computing y el Cloud Computing en base a las características que cada uno de ellos ofrece.

Estos intentos de realizar definiciones formales, sobre los conceptos concernientes a las tecnologías Cloud, ayudaron a empresas e investigadores a aclarar términos y a poder comunicarse utilizando los mismos conceptos.

Hasta el momento, todas las iniciativas relacionadas con el Cloud Computing asumíanla existencia de un proveedor externo de recursos en el que

se alojan los servicios y desde el que se ofertan las características del Cloud. Sin embargo, algunas empresas y muchos grupos de investigación ven en el Cloud Computing una serie de características interesantes, más allá de la utilización de gestores externos de servicios, como son la consolidación de servidores o la mejora en la eficiencia del uso de los equipos, que podrían aplicarse a los equipos ya disponibles. En este contexto surge el término "nube privada", como contraposición a las "nubes ofertadas al público" por parte de los proveedores de servicio (que no "nubes públicas").

Debido al interés despertado por esta perspectiva del Cloud Computing, surgieron una serie de proyectos dirigidos a permitir a los administradores crear una nube privada mediante la gestión conjunta de los recursos computacionales de la propia organización. Así, el término nube privada hace referencia a aquellas soluciones Cloud que no son ofrecidas por un proveedor y compartidas por los usuarios, sino que son habilitadas por una compañía para su utilización propia, o bien para convertirse a su vez en proveedores Cloud. Entre estas herramientas encontramos Nimbus [11] (Universidad de Chicago), Abicloud [29] (Abiquo), Open Nebula (Universidad Complutense de Madrid) y Eucalyptus (Universidad de California).

En la actualidad nos encontramos con que todas las grandes empresas del sector informático se han posicionado en esta tecnología y han adoptado las medidas necesarias para mantener una presencia activa en ella, al mismo tiempo que han aparecido una gran cantidad de proveedores de Cloud Computing en poco tiempo [30]. Asimismo, un buen número de grupos de investigación han reorientado su labor hacia esta tecnología emergente, en sus distintas facetas, abriendo nuevas líneas de investigación relacionadas con tecnologías Cloud o bien migrando aplicaciones científicas a entornos Cloud para aprovechar las posibilidades ofrecidas por estas soluciones.

En definitiva, podemos concluir que el Cloud Computing se perfila como una de las tecnologías más importantes en los años venideros en el sector de las IT. Esto es así tanto por el interés despertado en las empresas y el volumen de negocio estimado para los próximos años [31] como por las líneas de investigación a explorar en esta tecnología, que tienen el potencial de modificar los paradigmas de creación, distribución y consumo de software.

2.2. Estado del arte de las tecnologías asociadas

En esta sección se va a realizar una revisión del estado del arte de las principales tecnologías implicadas en el ámbito de la Tesis de Máster. Estas tecnologías han supuesto la base para el desarrollo del Cloud Computing y

en concreto del nivel de Platform-as-a-Service, por lo cual es preciso hacer una revisión de las mismas, para poder desarrollar el proyecto de Tesis.

2.2.1. Virtualización

El concepto de virtualización, en el contexto del Cloud Computing relativo a la infraestructura, hace referencia a la creación de una capa de abstracción entre el hardware de un computador y el software ejecutado sobre la misma. Esta tecnología oculta las características físicas de la máquina, ofreciendo en su lugar una máquina abstracta a los usuarios, y proporcionando un medio para crear una versión virtual de un dispositivo o recurso, potencialmente distinto al real. Para ello se basa en un elemento software que realiza la abstracción, y que se denomina "hipervisor".

La virtualización es una de las tecnologías básicas que han permitido el desarrollo de las tecnologías Cloud ya que, como hemos visto anteriormente, el Cloud Computing se basa en la provisión de una serie de recursos por parte de un proveedor a unos clientes. Precisamente esta tecnología permite ofrecer un nuevo modelo de provisión de recursos, realizando una virtualización de los mismos, y otorgando al Cloud Computing algunas de las características que le son propias. La virtualización aporta, por ejemplo, la homogeneización de los recursos para los clientes, independencia de la plataforma física utilizada y el particionado de los recursos físicos para no caer en restricciones relacionadas con el número de máquinas, al tiempo que introducen una capa adicional de seguridad y aislamiento.

Los primeros desarrollos en el campo de la virtualización datan de la década de 1960, cuando IBM buscaba una forma de dividir sus computadores *Mainframe* en particiones lógicas de menor tamaño. Estas particiones permitían al computador realizar multitarea, ejecutando múltiples aplicaciones y procesos al mismo tiempo, en una época en la cual estas y otras características no estaban tan extendidas como en la actualidad. Puesto que los Mainframes eran recursos caros, de esta forma se buscaba sacar el máximo rendimiento al hardware amortizando las inversiones realizadas.

Posteriormente, a lo largo las décadas de 1980 y 1990, la tecnología de virtualización se dejó a un lado, debido la aparición del paradigma cliente-servidor, la progresiva caída de precios de los servidores y la popularización de los computadores personales (PC). Todo esto hizo que no existiera la presión de los altos costes de inversión ni la necesidad de compartir los recursos tal y como se hacía con los Mainframes. Sin embargo, la amplia adopción de este tipo de soluciones estaba provocando que apareciesen progresivamente nuevos retos a afrontar a la hora de administrar las infraestructuras, como

los que se enumeran a continuación.

- Escasa utilización de las infraestructuras. El aislamiento de las aplicaciones por servidores y el hecho de que estas no estén siendo utilizadas continuamente provoca que los servidores trabajen a una fracción de su capacidad total, reportando una utilización promedio de entre el 15 % y el 30 % por servidor en un estudio realizado por IDC en 2005 [32], representando un valor en hardware desaprovechado de aproximadamente 140.000 millones de dólares en aquel momento. De hecho, esta tendencia se acentúa con el paso del tiempo, puesto que los avances del hardware proporcionan máquinas cada vez más potentes y, en muchos casos, el resultado es que se dispone de máquinas más desaprovechadas.

- Incremento del coste de la infraestructura. Con el aumento del número de servidores y del tamaño de los centros de datos destinados a albergarlos, se incrementan no sólo el coste debido a la adquisición de nuevas máquinas, sino también los costes de la infraestructura que los aloja. Adicionalmente al coste de nuevos servidores, armarios y espacios físicos para alojarlos, está el hecho de que la mayoría de máquinas deben permanecer encendidas de forma permanente, provocando unos costes de electricidad y refrigeración que son independientes del nivel de utilización del hardware.

- Incremento del coste de mantenimiento. Según aumenta la complejidad de los centros de datos, se requiere más personal especializado que realice las labores de mantenimiento. Un número creciente de servidores y la falta de automatización de ciertas operaciones requiere un aumento del personal, con el consiguiente aumento de costes.

- Coste de otros aspectos. Algunos aspectos adicionales como la seguridad, la disponibilidad, la robustez o la tolerancia a fallos son imprescindibles, en un entorno donde las empresas dependen cada vez más de los servidores para realizar operaciones críticas. Para afrontar estos retos, se toman decisiones que reducen la utilización de los servidores, o que resultan muy costosas, como son aislar los servicios por servidores, replicar servicios en varias máquinas o realizar copias de seguridad periódicas.

Estos retos llevaron a las empresas a plantearse volver al esquema de los grandes servidores y particionar el hardware en unidades lógicas, mediante técnicas de virtualización, para que puedan ser usadas por múltiples usuarios, y así mejorar la eficiencia en la utilización del hardware. Dentro de la

tecnología de virtualización podemos diferenciar distintas variantes, como la virtualización parcial, la paravirtualización y la virtualización completa.

- La virtualización parcial consiste en virtualizar algunos delos recursos hardware de una máquina, permitiendo a las aplicaciones hacer uso de estos recursos virtuales.

- La paravirtualización consiste en virtualizar todo el hardware de la máquina física. Sin embargo, existe un conjunto de operaciones sobre la máquina que no son virtualizadas, puesto que su ejecución está limitada al sistema operativo anfitrión. La aproximación de la paravirtualización a este problema consiste en incluir en el hipervisor un API a rutinas que implementan la funcionalidad de estas operaciones, y modificar el sistema operativo anfitrión para atrapar las operaciones no virtualizables y traducirlas a llamadas a este API, denominadas *hypercalls*. Una de las ventajas de la paravirtualización es que reduce el sobrecoste computacional asociado a las tecnologías de virtualización, si bien la necesidad de utilizar un sistema operativo modificado reduce la compatibilidad y portabilidad de la herramienta. Xen es un ejemplo de hipervisor que utiliza esta tecnología.

- La virtualización completa hace referencia a la simulación completa del hardware subyacente por parte de un hipervisor. La forma en la cual la virtualización completa aborda el problema de las operaciones no virtualizables es mediante la traducción de la misma a una secuencia de operaciones virtualizables que realizan la misma funcionalidad. Esta tecnología permite ejecutar cualquier software que se ejecute directamente sobre el hardware (es decir, sistemas operativos) sin modificación alguna, aunque como contrapunto genera un sobrecoste computacional al expandir el número de operaciones a realizar. VMWare es un ejemplo de hipervisor que utiliza esta tecnología.

Un sistema operativo que se ejecuta sobre una solución de virtualización se denomina máquina virtual, y este es el concepto que ha permitido el advenimiento del Cloud Computing. Para ello, se ha utilizado el particionado del hardware, mediante técnicas de virtualización, ejecutando múltiples instancias de sistemas operativos sobre una misma máquina física, ajustándose mejor a la carga asignada a cada máquina. Las ventajas que ofrece la virtualización para estos entornos son las siguientes.

- Mayor aprovechamiento de los recursos. Alojando varias máquinas virtuales en una máquina física se logra una mayor utilización del recurso,

y se deshecha el modelo de un servicio por servidor.

- Reducir los costes de infraestructura y administración. Con un mayor aprovechamiento de los recursos se reduce la necesidad de máquinas, y por ende de energía, refrigeración e infraestructura en general. Adicionalmente se necesitan menos administradores, y las propias máquinas virtuales facilitan las labores de mantenimiento.

- Aumentar la flexibilidad. Con el uso de máquinas virtuales se simplifican las operaciones de backup y migración de servidores, al tiempo que se obtiene la posibilidad de replicar o recuperar rápidamente un servicio, simplemente poniendo en marcha nuevas máquinas virtuales.

- Mejorar la seguridad. Al utilizar máquinas virtuales se ofrece a los usuarios entornos aislados entre sí y con la máquina física que los aloja. Así, en caso de que haya una brecha de seguridad en un servidor, no se compromete a la máquina física donde se aloja ni al resto de servidores.

- Tolerancia a fallos: Si un servicio que está alojado en una máquina virtual falla, el funcionamiento del resto de máquinas virtuales o de la máquina física anfitriona no resulta afectado. La máquina virtual puede ser reiniciada o intervenida de forma individual.

De forma adicional al punto de vista empresarial, un entorno virtualizado ofrece otras características interesantes para entornos de investigación o productivos.

- Homogeneización de recursos. Entornos físicos dispares pueden virtualizarse ofreciendo un entorno virtual homogéneo a los usuarios.

- Encapsulación. Una máquina virtual puede encapsular un entorno completo (de desarrollo, ejecución, test, etc.) y almacenarse para ser distribuida allá donde haga falta.

- Entornos de pruebas. Las máquinas virtuales pueden usarse como entornos de pruebas libremente, sin importar que se degraden con el uso. Cuando el entorno ya no resulta de utilidad, se puede eliminar la máquina virtual y desplegar de forma rápida y sencilla otro entorno nuevo, sin necesidad de modificar ninguna máquina física.

Todos estos argumentos apuntaban a que parte de la problemática, surgida en los nuevos centros de datos, podrían aliviarse con el uso de tecnologías

de virtualización. Y a pesar de que en 1999 la empresa VMware introdujo los primeros sistemas de virtualización completa para máquinas x86 con *"VMware Virtual Platform"* [33], esta tecnología no gozó de gran aceptación debido a que el hardware no estaba preparado para virtualización y suponía un sobrecoste computacional adicional.

En los años 2005 y 2006, Intel y AMD introdujeron en sus líneas de procesadores instrucciones de ayuda a la virtualización, las cuales simplificaban el proceso de virtualización completa, mediante las tecnologías Intel-VT [34] y AMD-V [35]. Estas tecnologías se siguieron desarrollando en sucesivas generaciones de procesadores, alcanzando progresivamente mayores niveles de rendimiento en los entornos virtuales. El resultado es que, en la actualidad, la tecnología de virtualización se ha desarrollado lo suficiente como para ser utilizada ampliamente en entornos empresariales y académicos.

La virtualización completa ha propiciado el desarrollo del Cloud Computing, principalmente del nivel Infrastructure-as-a-Service. Los proveedores Cloud de este tipo de soluciones utilizan las tecnologías de virtualización para realizar particiones lógicas de grandes centros de datos en unidades de tamaños predeterminados. Utilizando este modelo, los proveedores optimizan la utilización de los recursos físicos, maximizan el número de clientes al que se puede servir y simplifican las labores de administración. Desde el lado de los usuarios, éstos obtienen la posibilidad de utilizar recursos computacionales bajo demanda sin la necesidad de adquirir y mantener nuevo hardware, adoptando un modelo de pago por uso.

Dentro del contexto de la Tesis de Máster, se ha realizado un estudio de las tecnologías de virtualización para analizar cómo estas permiten aportar características cloud (aislamiento, particionado, elasticidad, etc.). La intención es la de poder aplicar los conceptos de virtualización de máquinas a la virtualización de servicios.

A partir de este estudio, y para dar soporte a los servicios virtuales, se ha planteado el concepto de "Contenedor de Servicios Virtual", o *Virtual Container*. El *Virtual Container* es un elemento análogo al hipervisor de la virtualización hardware, trasladado al contexto de los Servicios. Mediante la utilización de *Virtual Containers*, una plataforma PaaS proporciona un *runtime* virtual donde los usuarios pueden alojar y ejecutar su software, abstrayéndose del hardware subyacente. La utilización de este componente como medio para la virtualización del entorno de ejecución de servicios permite proporcionar a las plataformas PaaS unas características análogas a las que proporcionan las plataformas IaaS.

2.2.2. Servicios web

Un servicio web es un sistema software diseñado para soportar operaciones entre máquinas a través de la red [36]. Los servicios web exponen sus funcionalidades a través de un interfaz descrito en un formato procesable automáticamente (WSDL [37]), y otros sistemas interactúan con el servicio web siguiendo esta especificación usando mensajes codificados en un formato estándar (SOAP [38]), típicamente transmitidos a través de HTTP codificados en XML.

Los servicios web se componen de dos elementos que intercambian información: los servicios y los clientes. Los servicios representan el elemento pasivo del sistema, mientras que los clientes representan el elemento activo, que es el que inicia las comunicaciones.

Para realizar una comunicación, el cliente codifica la interacción (el intercambio de mensajes que desea efectuar) que desea llevar a cabo (por ejemplo llamada a un procedimiento junto con los parámetros requeridos) en un lenguaje estándar, como SOAP. En particular SOAP es una especificación propia de los servicios web, que define el formato de la información estructurada que intercambian clientes y servidores, y que se codifica mediante el lenguaje de marcas XML. Un mensaje SOAP se compone de un sobre que incluye unas cabeceras, que aportan metadatos sobre el mensaje, como directrices para su procesamiento, y un cuerpo, que incluye la información a intercambiar.

Una vez se ha construido el mensaje SOAP, este puede opcionalmente procesar mediante una capa de middleware, que eventualmente realizará modificaciones sobre el mensaje, en caso de que sea necesario, como ocurre cuando se utilizan algunas de las recomendaciones WS-*. Las recomendaciones WS-* son un conjunto de especificaciones que abordan distintos aspectos de los servicios web, como son la seguridad, el intercambio de metadatos, la identificación de servicios, etc.

En último lugar, el mensaje SOAP procesado se envía al servicio a través de algún protocolo, típicamente HTTP, aunque puede utilizar otros como HTTPS, FTP, JMS, etc.

En el lado del servidor, un *Transport Listener* atiende los mensajes entrantes del protocolo que soporta (es decir, un *HTTP Listener* atiende mensajes HTTP), lo despacha a un middleware que procesa el mensaje en base a las directivas indicadas en las cabeceras del mismo y en último lugar delega al servicio web el procesamiento de las operaciones indicadas en el cuerpo del mensaje.

El objetivo de los servicios web es proporcionar a los clientes un nuevo

paradigma para el consumo de servicios, como resultado de la evolución del modelo cliente-servidor tradicional. La forma en la cual se utilizan los servicios web para ofrecer servicios a los clientes da lugar a distintos estilos de uso, pero los más comunes se describen a continuación.

- Remote procedure calls: Los servicios web pueden ser utilizados para soportar llamadas a procedimientos remotos. En este modelo, los servicios web exponen el interfaz de sus operaciones al exterior, y se utiliza SOAP como envoltorio para las llamadas al servicio. Mediante un proxy, los clientes pueden realizar llamadas al servicio web como si fuese una llamada local.

- Service Oriented Architecture: Los servicios web pueden utilizarse para implementar una arquitectura orientada a servicios. Este tipo de arquitecturas, descritas con detalle en el apartado x.x 2.1.3, se basan en la interacción mediante mensajes de servicios débilmente acoplados. En una implementación de este tipo de arquitecturas usando servicios web, es ésta la tecnología que implementa los servicios, y SOAP el lenguaje utilizado para el intercambio de mensajes.

- Representational State Transfer: Representational State Transfer, o REST, es un estilo de arquitectura software orientada a sistemas de hipermedia distribuidos (por ejemplo World Wide Web). El objetivo principal de las arquitecturas REST es el de proporcionar una serie de características deseables para sistemas distribuidos (escalabilidad, fiabilidad, eficiencia, simplicidad, portabilidad) mediante la imposición de una serie de restricciones sobre la arquitectura. Estas restricciones son las siguientes.

  - Modelo cliente-servidor: El sistema se divide en componentes clientes y servidores, separando el interfaz de usuario de los contenidos.
  - Interacciones sin estado: Una interacción contiene toda la información necesaria para servir una petición, sin necesidad de tener conocimiento de interacciones anteriores.
  - Caché: Las interacciones pueden ser cacheadas por el cliente. Es responsabilidad de cada interacción indicar explícitamente si es o no cacheable.
  - Interfaces uniformes: Todos los componentes interactúan a través de interfaces uniformes con operaciones bien definidas. Estos interfaces a su vez están sometidos a una serie de restricciones.

    - Identificación de recursos: Los recursos con los cuales se pretende interactuar son referenciados de forma explícita en las peticiones a los servidores. Los recursos están identificados de forma univoca en el sistema (por ejemplo mediante una dirección URI).
    - Manipulación de recursos a través de su representación: Cuando un cliente posee una representación de un recurso (es decir, la forma en la cual un recurso ha sido representado en el sistema, incluyendo sus metadatos) esta información debe ser suficiente como para que el cliente pueda manipular el recurso mediante las operaciones del interfaz.
    - Mensajes descriptivos: Los mensajes intercambiados entre elementos poseen la información suficiente para indiciar a los componentes cómo procesarlo.
    - Hipermedia como motor de los cambios de estado: Cuando se le proporciona un recurso a un cliente, los otros recursos referenciados en el mismo deben estar representados como hipermedia, de tal forma que se permita al cliente transitar entre estados utilizando las representaciones.
- Sistema por capas: Los componentes de la arquitectura se pueden organizar en capas y generar cadenas de interacciones, de forma transparente para los extremos de la misma. Por ejemplo, un cliente puede enviar una petición a un servidor a través de un proxy sin la necesidad de conocer la existencia del mismo.
- Código bajo demanda: El servidor puede delegar parte de la funcionalidad al cliente en forma de código ejecutable (por ejemplo scripts). Esta es la única característica que se define como opcional.

Los web services pueden implementar una arquitectura REST basándose en las operaciones HTTP, de forma que un servicio web RESTful es aquel que se ajusta a las restricciones REST. Estos servicios ofrecen un interfaz uniforme a través de la utilización de las operaciones de HTTP GET, PUT, POST, DELETE, etc. realizadas sobre URIs que representan recursos en el servicio. Estos servicios web pueden implementarse realizando una abstracción de las operaciones sobre SOAP o utilizando HTTP de forma directa.

Las ventajas de los servicios web, sobre el modelo cliente servidor tradicional son básicamente las siguientes:

- Interoperabilidad: Puesto que las comunicaciones entre servicios web se realizan mediante lenguajes estandarizados, un cliente puede comunicarse con un servicio con independencia del lenguaje, sistema operativo, etc.

- Modularidad, diseño desacoplado: Los servicios web encapsulan funcionalidad ofrecida a través de interfaces, con independencia de la implementación. Esta modularidad permite cambiar las implementaciones sin modificar el software que utiliza el servicio. La implementación modular también facilita el proceso de desarrollo.

- Reusabilidad: Si existe una funcionalidad básica, que va a ser utilizada por múltiples servicios, puede implementarse una vez y ser reutilizada en subsiguientes desarrollos.

- Combinación de software: Puesto que los servicios ofrecen funcionalidades de forma abierta a todos los usuarios, nuevos desarrollos pueden utilizar dicha funcionalidad para la implementación de otros servicios, aun cuando el objetivo original del servicio web utilizado no era ese.

Debido a la popularidad de esta modalidad de distribución de software, a sus características y a la flexibilidad que presenta, los servicios web suponen una aproximación adecuada para la provisión de servicios en un entorno Cloud. En el nivel PaaS del Cloud, el recurso ofrecido es un *runtime* virtual, mientras que el usuario es el desarrollador que utiliza este recurso para alojar y ejecutar su software. Por tanto, a la hora de definir una solución PaaS, es necesario especificar a qué tipo de software se va a dar soporte y qué *runtime* se va a virtualizar.

La utilización de servicios web como base para Cloud ComPaaS, permite ofrecer un *runtime* que dé soporte a servicios independientemente de su lenguaje y estilo (RPC o REST), que pueden ser utilizados para realizar labores de cálculo o bien para servir peticiones de usuarios externos a través de un interfaz público. Adicionalmente, la implementación de los servicios propios de la plataforma (*core services*) como servicios web mantiene la uniformidad y coherencia del sistema. El *runtime* que aloja los servicios web se denomina web service container. Estas herramientas habilitan el proceso de interacción de cliente y servicio, en base al esquema expuesto anteriormente. Cuando se diseña una plataforma PaaS, es necesario elegir a qué tipo de software se va a dar soporte en la misma, es decir, qué tipo de *runtime* se va a virtualizar y qué lenguajes o características van a poder utilizarse en el entorno que proporciona.

En el ámbito de la Tesis de Máster, hemos seleccionado los web services como los servicios de usuario que van a poder ser desplegados en la plataforma. Asimismo, la propia implementación de Cloud ComPaaS se va a realizar como servicios web, y el *runtime* virtual que conforma la solución va a consistir en contenedores de servicios virtuales, o *Virtual Containers*, aplicaciones adaptadas para proporcionar las características de las tecnologías de virtualización a nivel de servicio.

2.2.3. Service Oriented Architecture (SOA)

Una arquitectura orientada a servicios (SOA) es un conjunto de patrones de diseño a utilizar durante el desarrollo de un sistema, con el objetivo de proporcionar unos ciertos beneficios sobre otro tipo de arquitecturas software. Estos patrones son básicamente los siguientes

- Modularidad: La lógica del sistema se divide en módulos bien definidos que implementan operaciones sencillas. Estos módulos se encapsulan en componentes software denominados servicios.

- Granularidad y relevancia: Los servicios implementan funcionalidades en una granularidad suficiente como para resultar relevantes para el sistema y los usuarios. Este principio evita el diseño de servicios demasiado complejos o demasiado simples.

- Reusabilidad: Los servicios se diseñan con el objetivo de propiciar la usabilidad y evitar la duplicación de código entre los distintos módulos La reusabilidad se basa en el principio de granularidad y en la relevancia de los servicios.Publicación: Cada uno de los servicios que componen el sistema debe publicar información precisa sobre su funcionalidad a través de un interfaz.

- Abstracción: El interfaz de los servicios, que está expuesto públicamente, es una abstracción de la implementación del mismo. Este principio implica que los interfaces de los servicios deben ser genéricos y ocultar los detalles de implementación subyacentes.

- Acoplamiento débil: Cada uno de los servicios que componen el sistema deben ser independientes entre sí o estar débilmente acoplados. Este principio implica que el funcionamiento de un servicio se realiza en base a interfaces genéricos y no a los detalles de implementación del mismo. El acoplamiento débil se beneficia del principio de abstracción.

- Composición: Varios servicios alojados en el sistema pueden ser utilizados de forma conjunta para componer servicios de complejidad creciente. La composición de servicios se beneficia del principio de acoplamiento débil.

- Descubrimiento: El sistema incluye un mecanismo que permite descubrir los servicios alojados en el mismo. El sistema de descubrimiento proporciona información, el interfaz y un método para acceder y utilizar el servicio.

- Uso de estándares: Todas las comunicaciones e interacciones en el sistema se llevan a cabo utilizando estándares, propiciando la interoperabilidad e independencia de la plataforma.

Por tanto, en base a estos principios, las SOA se implementan como un conjunto de servicios débilmente acoplados, donde cada uno implementa una funcionalidad básica y se comunican mediante intercambio de mensajes. Para habilitar este intercambio de mensajes, los servicios deben disponer de metadatos que indiquen las interfaces de los otros servicios, así como el formato de los datos, patrones de intercambio, etc. Adicionalmente, las SOA incluyen un sistema que permite buscar y utilizar los servicios en base a sus metadatos e interfaces.

En la definición de SOA, el concepto de servicio se enuncia como un componente software que posee una serie de características descritas en los principios de diseño. Si bien cualquier componente que cumpla con dichos requisitos es un candidato válido a implementar una SOA, debido a sus características los servicios web se han posicionado como el candidato idóneo para desempeñar este papel.

Los conceptos de *web services* y SOA fueron desarrollados de forma independiente, aunque su estrecha relación ha producido que el SOA se haya definido como una de las aplicaciones de los web services, y que los web services se hayan designado como requisito para la implementación de un SOA. Si bien los SOA tratan con servicios abstractos, ciertamente los web services proporcionan en su especificación un conjunto de características que forman parte de los patrones de diseño SOA. Algunas de estas características, expuestas en el apartado x.x 2.1.2 son la modularidad, la reusabilidad, la composición de servicios y el uso de estándares. Adicionalmente, la capacidad de los web services para describir su interfaz en un formato estándar (WSDL) y la existencia del estándar UDDI, facilitan la labor de publicación y descubrimiento de servicios.

Sin embargo, a pesar de que los servicios (web services en particular) forman parte fundamental del concepto de SOA, no definen la amplitud de la definición del mismo. Los servicios representan un interfaz a una funcionalidad que proporciona una serie de características deseables, pero en sí no definen la arquitectura del sistema o cómo estos componentes actúan y se relacionan. Por tanto el objetivo del SOA es proporcionar un nuevo modelo de desarrollo de sistemas software complejos que recopila un conjunto de buenas prácticas que permite disminuir el tiempo y coste de desarrollo, aumentar la calidad del producto software, aumentar las capacidades de interoperabilidad y federación de los sistemas, además de proporcionar a los consumidores funcionalidades relevantes y de fácil uso.

Las SOA se perfilan como un modelo de oferta de servicios prometedor en el campo de los negocios. Es por ello que muchas grandes empresas ofrecen SOA como producto empresarial [39], mientras que otras ya han comenzado a relacionar esta forma de proveer servicios con el Cloud Computing [40].

En el contexto de la Tesis de Máster, la arquitectura de Cloud ComPaaS se ha definido en base a los principios SOA. Esta decisión es una consecuencia de la elección de los web services como medio para la implementación de los componentes del sistema. Puesto que el nivel PaaS del Cloud Computing está orientado al alojamiento, ejecución y provisión de servicios, resulta natural utilizar uno de los estilos de uso de los servicios (mencionados en el apartado x.x 2.1.2) para el diseño de una plataforma PaaS. Utilizando una aproximación SOA, no sólo otorgamos a la plataforma todos los beneficios de estos principios, sino que podemos extenderlos a los servicios que a su vez van a ser alojados en el sistema.

Capítulo 3

Especificación de la plataforma y diseño de módulos

En este apartado se recoge la especificación de la plataforma PaaS, en base a los requerimientos y características esperadas para este tipo de soluciones. Para establecer dichas características nos hemos basado tanto en el estudio de las plataformas comerciales de este tipo, como en los proyectos y artículos de referencia de la materia, introducidos a lo largo del estado del arte. Además se recogen opiniones vertidas por usuarios y diseñadores en blogs, además de la experiencia previa en temas de Grid Computing y computación paralela. Junto con la especificación de la plataforma se plantea el diseño de una arquitectura que da respuesta a dichos requerimientos.

Para estructurar el desarrollo del trabajo, hemos planteado la especificación de la plataforma Cloud de acuerdo a las siguientes fases:

1. Especificación de las características deseadas para la plataforma objetivo.

2. Diseño de una arquitectura que integra y comunica los elementos que proporcionan dichas características.

3. Definición de los módulos que componen la arquitectura, indicando para cada uno su funcionalidad, interfaces y patrones de comunicación.

El resultado de este proceso es la especificación completa, con su arquitectura, módulos y patrones de interacción de la plataforma. Esta especificación se tomará como base para la implementación de una plataforma Cloud que posee las características deseadas.

A lo largo de este capítulo se exponen los tres pasos de la especificación realizada para Cloud ComPaaS.

3.1. Visión general de la plataforma

El objetivo de la plataforma es el de ofrecer a los desarrolladores un entorno de ejecución donde alojar y ejecutar sus servicios, y permitir a los consumidores buscar servicios alojados y obtener acceso a los mismos.

En el arranque de la plataforma, los administradores deberán poner en marcha una serie de servicios de la plataforma, que denominamos *core services*. Cada uno de los *core services* posee unas credenciales de cloud, con las cuales se autentifica con el resto, y al mismo tiempo formarán un clúster en el cual todos los nodos se conocen y pueden comunicarse de forma interna, segura y fiable.

Estos *core services* podrán tener operaciones públicas o privadas. Aquellos que tienen operaciones públicas (para la búsqueda de servicios, despliegue de servicios, etc.) presentan una descripción de sus operaciones, que además podrán ser accedidas desde el exterior. Los servicios que dispongan de operaciones privadas, las presentan mediante un interfaz que conecte con un sistema interno de comunicaciones, mediante el que poder intercambiar información entre los elementos de la nube, siguiendo un paradigma de paso de mensaje.

Las operaciones públicas de la plataforma deberán ser accesibles para los usuarios que se encuentran registrados en la misma. Las capacidades de los usuarios registrados (qué operaciones pueden realizar) vendrán determinadas por una serie de roles de usuario. De forma general de los tipos de usuarios que existen en el sistema es el siguiente:

- Los usuarios de rol administrador tendrán la capacidad de acceder a las operaciones de administración de la plataforma. Estas operaciones incluyen administración de usuarios, modificación de las reglas de elasticidad, modificación de las reglas de interoperabilidad y, en general, todas aquellas operaciones que modifiquen el comportamiento de la plataforma.

- Los usuarios de rol desarrollador deberán tener la capacidad de alojar servicios en la plataforma, desplegar instancias de sus servicios y administrar estos elementos. Los usuarios desarrolladores tendrán asociada un SLA que deberá incluir los términos de uso de la plataforma. Este

SLA será un documento en el que se definan los recursos a los que este tiene acceso a la hora de desplegar instancias de servicios.

- Los usuarios del rol consumidor tendrán la capacidad de ejecutar operaciones públicas de la plataforma, como la consulta al catálogo, además de poder utilizar los servicios que pongan a su disposición los desarrolladores.

Para poner en marcha instancias de servicios, en primer lugar el desarrollador deberá registrar los servicios en el sistema. El procedimiento de registro consiste en proporcionar a la plataforma los metadatos y el paquete binario del servicio web que se desee poner a disposición de los usuarios. Uno de estos metadatos será el SLA, donde se definirán requisitos y restricciones de ejecución. Los servicios registrados se almacenan en un sistema de información, para que sus datos puedan ser consultados públicamente y el paquete binario pueda ser recuperado por el resto de componentes.

Una vez un servicio se ha registrado en el sistema, el desarrollador puede desplegar instancias del mismo. El despliegue de una instancia consiste en ejecutar un servicio en la plataforma, siguiendo las directrices marcadas por el SLA que el desarrollador define para la misma. En caso de que los SLAs sean válidos y se permita al desarrollador desplegar la instancia, la plataforma elige un *runtime* virtual donde ejecutar la instancia en base a las restricciones impuestas por el propio SLA y la utilización de un algoritmo de equilibrado de carga.

En caso de que no haya *runtimes* libres para desplegar la instancia, la plataforma procederá a desplegar nuevos entornos virtuales en la infraestructura física disponible, utilizando las credenciales proporcionadas por los administradores para acceder a las máquinas, y utilizando algoritmos de equilibrado de carga para asignar los recursos computacionales de forma optimizada.

En caso de que se acaben los recursos físicos donde alojar nuevos nodos, la plataforma acudirá a otras nubes, utilizando reglas de interoperabilidad, para solicitarles recursos. Mediante un proceso de negociación, el despliegue puede alquilar recursos a otra nube y añadirlos a su infraestructura de forma temporal.

Cuando una nueva instancia se pone en marcha, sus datos se registran en un sistema de información. Uno de los datos de la instancia es su "endpoint reference", una dirección URI que representa su localización en la nube. Una vez desplegada, cualquier cliente autorizado a ello puede acceder a ella a través de internet. Para ello, los clientes necesitan disponer de una "endpoint reference", la cual deben conocer de antemano, o podrán obtener mediante los mecanismos de búsqueda de servicios de la plataforma.

Para obtener una "endpoint reference" de una instancia de un servicio, los consumidores tienen dos opciones. En primer lugar pueden solicitar a la plataforma una "endpoint reference" de un servicio identificado por su nombre, que seleccionará una de entre todas las instancias de ese servicio en base a un algoritmo de equilibrado de carga. En segundo lugar, pueden consultar de forma directa el sistema de información para obtener la "endpoint reference" de una instancia concreta de un servicio, o bien obtener una colección completa de ellas. Una vez accedido a la instancia, el usuario podrá disponer de mecanismos de almacenamiento en el cloud, ligados a su identidad y a las instancias del servicio que ha accedido. Este espacio de almacenamiento estará aislado del almacenamiento de otros usuarios, aun cuando hayan accedido a la misma instancia del servicio.

Todas estas operaciones describen el comportamiento de la plataforma frente a la acción de usuarios externos. Sin embargo la propia plataforma deberá realizar una serie de tareas de forma autónoma orientadas a mejorar su funcionamiento. Entre este tipo de funcionalidades se encuentran la monitorización del sistema, la auto recuperación y el control de la calidad de servicio.

El sistema de monitorización se encuentra integrado en todos los componentes de la plataforma, y su cometido es el de realizar medidas sobre distintos aspectos y elementos de los mismos, y registrar esta información en el sistema de información de la plataforma. La información de monitorización se puede dividir en dos categorías en función de la naturaleza de la misma: las estadísticas, que representan información caracterizada a lo largo del tiempo (por ejemplo el número de usuarios que han accedido a una instancia desde que comenzó a ejecutarse) y las métricas, que representan información puntual que pierde sentido transcurrido un tiempo (por ejemplo número de usuarios que están accediendo a una instancia en el instante actual). Cada uno de los servicios deberá estar instrumentado para recopilar la información de monitorización que se considere relevante y enviarla de forma periódica al sistema de información para su registro.

La plataforma también trabaja de forma autónoma para asegurar la robustez del sistema, manteniendo un registro con los nodos que conforman el despliegue, denominado configuración. Esta configuración es proporcionada por los administradores y será modificada de forma dinámica según se despliegan o repliegan nuevos componentes. Cuando la plataforma detecta que un nodo ha abandonado el sistema (gracias al sistema de clustering) comprobará de forma automática si el nodo pertenece a la configuración, es decir, si el nodo ha sido replegado de forma activa o si ha fallado. En caso de fallo, el componente procede a eliminar el nodo fallido y desplegar uno

nuevo para que ocupe su lugar.

En último lugar la plataforma se encarga de realizar comprobaciones periódicas sobre la calidad de servicio que ofrece a los clientes. Los mecanismos que utilizará para determinar este concepto son básicamente dos: los SLAs y los indicadores de calidad de servicio. La plataforma comprobará de forma periódica y continua los SLAs de los distintos elementos que la componen, verificando que todos ellos son válidos. En caso de que algún SLA haya dejado de ser válido (por ejemplo una instancia ha fallado y ha dejado de ejecutarse) se llevarán a cabo acciones correctoras, orientadas a solucionar la situación problemática y restablecer el SLA. Estos eventos son registrados en la plataforma para poder notificar a los usuarios implicados, y eventualmente realizar las compensaciones que puedan haber sido acordadas. Los indicadores de calidad de servicio consistirán en un conjunto de parámetros utilizados para medir la calidad general del servicio que está prestando la plataforma (por ejemplo tiempo de respuesta, carga de los componentes, etc.). En base a estos parámetros se definirán unas reglas de elasticidad que determinarán cuándo se deben desplegar o replegar componentes en la plataforma, o cuándo se debe aumentar o disminuir el número de réplicas de un servicio.

Todos estos elementos componen la visión general del funcionamiento de la especificación realizada para Cloud ComPaaS, abarcando un amplio abanico de funcionalidades y áreas del conocimiento. Esta especificación se ha guiado por unos principios de diseño, y se ha plasmado en una arquitectura y en el diseño de los módulos que dan soporte a la misma, que van a ser descritos en los siguientes apartados.

3.2. Principios de diseño

En este apartado se describen en detalle los principios de diseño de la plataforma, exponiendo para cada uno de ellos su motivación en el contexto del Cloud Computing, y las características que aporta a la plataforma.

3.2.1. Administración de servicios

El objetivo principal de una nube PaaS consiste en ofrecer un *runtime* virtual donde se pueden alojar y ejecutar servicios, de la misma forma que una nube IaaS ofrece máquinas virtuales. Por tanto, para habilitar el despliegue de nubes PaaS es necesario ofrecer un sistema que permita administrar dichos servicios virtualizados.

Para lograr este objetivo, el sistema define unos componentes lógicos denominados contenedores virtuales, que desempeñan a nivel de servicio el

papel de los *hipervisores* en la virtualización hardware, creando *runtimes* virtuales donde los desarrolladores pueden alojar y ejecutar sus servicios.

De esta forma, se pretende centrar el diseño de los componentes en torno al concepto de contenedor virtual. Por tanto, el diseño de dichos componentes reflejará la gestión y administración de servicios virtuales, y se logra generar un sistema que implementa una nube PaaS de acuerdo a las premisas planteadas.

3.2.2. Multitenancy

En un entorno virtualizado se generan particiones lógicas de un hardware y se asigna cada partición a un usuario distinto. Sin embargo, la realidad es que el hardware subyacente está siendo compartido por los distintos usuarios. En este contexto, la característica de *multitenancy* consiste en proporcionar la capacidad de que un mismo recurso pueda servir a distintos usuarios (tenants), pero de tal forma que cada uno de estos usuarios esté aislado del resto y no se interfieran entre sí.

Una de las características esperadas de los *runtimes* virtuales es que proporcionen la impresión de ser entornos dedicados, equivalentes a entornos reales. Cuando un usuario realiza una petición a la nube, recibe un elemento virtualizado, aislado del resto, y preparado para su uso exclusivo. Dar acceso a elementos compartidos no aislados permitiría a los usuarios interactuar con material perteneciente a otros de forma trivial, generando riesgos relativos a la seguridad.

La característica de multitenancy permite, en nuestro caso, que la plataforma asigne los mismos recursos físicos a varios usuarios. Utilizando esta aproximación se consigue una optimización en la utilización de los recursos y se mejora la escalabilidad y el rendimiento de la plataforma, puesto que distintos consumidores pueden compartir la misma infraestructura física.

3.2.3. Quality of Service (QoS)

En una plataforma distribuida y compartida por varios usuarios se debe proporcionar de forma transparente mecanismos que aseguran el nivel de calidad de servicio (QoS), robustez, rendimiento y optimización de la utilización de recursos, con el objetivo de poder proporcionar a los usuarios una experiencia satisfactoria, de acuerdo a las características que hayan solicitado. Algunos ejemplos de los mecanismos que permiten ofrecer estas características son el equilibrado de carga, la replicación de elementos y el auto escalado de servicios en la plataforma, para satisfacer aumento en la

demanda.

Para proporcionar robustez se pueden seguir una aproximación basada en la replicación y de este modo desplegar múltiples instancias de un mismo servicio, de forma simultánea. De esta forma, en caso de que alguna de dichas instancias deje de funcionar, los usuarios continuarían teniendo acceso al servicio.

Por su parte, el equilibrado de carga puede mejorar el rendimiento, evitando la saturación de los recursos y mejorando el tiempo de respuesta. Esto se puede lograr repartiendo el acceso de los usuarios de la plataforma entre los recursos distribuidos, siguiendo un cierto criterio a dos niveles. Por un lado, las instancias a desplegar se pueden dividir entre los recursos disponibles, mientras que por otro las peticiones de los usuarios a los servicios en sí se pueden distribuir entre las distintas instancias de los servicios.

La gestión automática del escalado de recursos (denominada auto escalado en nuestro contexto) permite optimizar la utilización de los recursos, adecuando la disponibilidad de estos a la carga actual del sistema. Mediante el escalado de aumento (*upscaling*) se evita la carencia de recursos que produce un rendimiento pobre, mientras que mediante el escalado de disminución (*downscaling*) se evita el exceso de recursos disponibles en la plataforma, lo cual produce desperdicio de recursos y sobrecoste económico.

En una nube PaaS el escalado se puede considerar a dos niveles: los servicios y la plataforma en sí. A nivel de servicio, son las instancias de las aplicaciones las que escalan en número en base a unas reglas de elasticidad definidas por los desarrolladores de las mismas. Por otro lado, a nivel de plataforma, son los propios componentes del sistema los que deben escalar, basándose en la carga del propio sistema y en los recursos físicos disponibles.

Ambas variantes del auto escalado son similares, pero cada una requiere de una aproximación propia en cuanto a las reglas de elasticidad, equilibrado de carga, protocolos de escalado (despliegue y repliegue de elementos), etc.

3.2.4. Monitorización, Contabilidad y Facturación

Una de las características básicas del Cloud Computing es el modelo de pago por uso, en el cual a los clientes se les cobra únicamente por los recursos que han utilizado. Para poder proporcionar este modelo de pago, es necesario disponer de mecanismos que registren la utilización de recursos por parte de los servicios y de los usuarios, en base a los cuales realizar la contabilidad y facturación.

De forma general, la contabilidad es la acción de registrar, clasificar y resumir los eventos que tienen una relevancia financiera en el sistema. Por

su parte, la facturación es el proceso por el cual se controla el pago de los recursos ofrecidos, tal y como ha registrado la contabilidad. De acuerdo a estas definiciones, podemos interpretar que para poder ofrecer un modelo de pago por uso es necesario disponer de mecanismos que midan la utilización de los recursos y que controlen el pago por la utilización de los mismos.

Las labores de mantenimiento general de la plataforma también se benefician de la monitorización del sistema. Mediante una herramienta de monitorización se puede ofrecer a los administradores información actualizada sobre los componentes y los servicios, al tiempo que se habilita, de forma sencilla, la comprobación de los parámetros de Calidad de Servicio, la carga de los elementos, el estado de los componentes, etc.

La monitorización de la plataforma se logra mediante una aproximación "*push*", en la que los elementos aportan información de forma activa. Cada uno de los componentes realiza medidas sobre su estado de forma periódica, empaqueta los datos en un informe y lo envía al catálogo, que es el componente encargado de almacenar la información del sistema.

Algunas de las medidas realizadas registran la carga de los elementos, tal como el uso de CPU, ancho de banda, almacenamiento, número de peticiones en curso, etc. Por otro lado, se realizan otras medidas más complejas, con la intención de poder ofrecer información más detallada, como el tiempo de respuesta promedio, el histórico de uso, etc.

3.2.5. Seguridad, privacidad, control de acceso

Una solución cloud destinada a la provisión de recursos virtualizados debe disponer de un marco de seguridad que le permita identificar usuarios, asegurar la privacidad de los datos y controlar el acceso al sistema. Este marco de seguridad debe evitar el uso inapropiado de la plataforma, sus recursos y los servicios alojados en ella, tanto a nivel de la plataforma en sí, como por parte del acceso a los servicios. De esta forma, la plataforma debe contemplar mecanismos de autenticación que permitan autenticar e identificar a los usuarios. Este mecanismo se puede implementar finalmente utilizando métodos estándar, incluidos en librerías públicas, como username-token, certificados X509, tokens SAML, etc.

A partir de esta identificación se define el patrón de interacción entre los usuarios y la plataforma, mediante técnicas de control de acceso. Estos patrones especifican, por ejemplo, qué usuarios pueden administrar la plataforma, desplegar servicios, utilizar aplicaciones desplegadas, etc.

Finalmente, se deben establecer unos roles de usuario que definan la tipologías de usuarios que existen en la plataforma, las capacidades que tiene

cada uno, la información asociada a los mismos y la forma en la cual esta se almacena y accede. Se pueden considerar distintos roles, como por ejemplo un rol Desarrollador, que puede registrar servicios pero no administrar la plataforma, otro destinado a consumidores de servicios, etc.

3.2.6. Almacenamiento Cloud

El almacenamiento es uno de los puntos sensibles de una solución Cloud, y su relevancia resulta aún mayor en un entorno PaaS debido a que el tratamiento de este determinará una serie de características adicionales de la plataforma. En cualquier caso, el almacenamiento es un recurso, y como el resto de recursos (CPU, Memoria, etc.), debe ser virtualizado. La virtualización del almacenamiento se realiza interviniendo en todas las operaciones relacionadas con el mismo, proporcionando primitivas orientadas al almacenamiento Cloud.

El uso de una solución Cloud requiere la utilización de almacenamiento Cloud, pues debe cumplir con unas características que no incluyen los sistemas de almacenamiento tradicionales. Este tipo de almacenamiento proporciona un sistema de ficheros virtual (alternativamente, una base de datos virtual) que permite a los usuarios almacenar y recuperar información independientemente de la implementación subyacente.

Un sistema de ficheros virtual, junto con el sistema de autenticación, enlaza un espacio de almacenamiento virtual con un usuario, y proporciona características no disponibles con las primitivas de almacenamiento tradicionales, como son las siguientes:

- Trazado: Los sistemas de ficheros convencionales no proporcionan soporte para el trazado de la actividad y es por ello que requieren de una capa adicional que le dotará de estas las características. Sin embargo un sistema de ficheros virtual atrapa todas las operaciones de almacenamiento y por tanto las puede registrar. Este registro permite a la plataforma monitorizar el recurso de almacenamiento, permitiendo establecer nuevas medidas como espacio ocupado, número de ficheros, tamaño de ficheros, etc. Estas medidas permiten limitar el uso del almacenamiento por parte de los usuarios.

- Aislamiento: Los usuarios reciben la visión de un (inicialmente vacío) sistema de ficheros al cual solamente ellos tienen acceso. Los usuarios no pueden interactuar con el espacio de otros o con el sistema de almacenamiento físico subyacente, mejorando la seguridad.

- Ubiquidad: Uno de los principios Cloud es que las peticiones de clientes de un recurso puedan ser atendidas por cualquier servidor, favoreciendo la escalabilidad y tolerancia a fallos. Este principio obliga a que las interacciones no impliquen un estado, pues en caso de mantener estado en las interacciones obligaría a los clientes a dirigir todas las peticiones al mismo servicio. En base a este requisito, en caso de querer ofrecer almacenamiento en los servicios, es necesario que este sea ubicuo para asegurar que todas las instancias tienen acceso al mismo.

- Replicación y distribución: La utilización de un sistema de ficheros virtual permite la distribución y replicación de la información de forma transparente para los usuarios, dándoles una imagen de un sistema de ficheros tradicional. La replicación y distribución puede incrementar la QoS, disminuyendo el tiempo de acceso al permitir el equilibrado de carga en operaciones de lectura, y mejorar la robustez, previniendo la pérdida de información en caso de fallos.

- De forma complementaria es necesario proporcionar mecanismos que permitan a los usuarios especificar el grado de replicación deseado para los datos o el almacenamiento de información en una ubicación física específica.

3.2.7. Service Level Agreement (SLA)

En un entorno virtual donde potencialmente se pueden realizar transacciones comerciales, como es un entorno cloud, es importante definir mecanismos que den garantías legales a las partes implicadas sobre los servicios que se van a prestar. En el ámbito del software se ha adoptado el Service Level Agreement (SLA) como medio para plasmar este tipo de acuerdos.

Un SLA es un contrato legalmente vinculante entre un proveedor y un usuario, que define los términos del acuerdo entre ambas partes a nivel de servicio, especificando obligaciones, responsabilidades, garantías y compensaciones. Si bien este contrato se puede definir de cualquier manera mientras los clientes sean capaces de entenderlo y aceptarlo, comúnmente se ofrece simplemente como un documento de texto que es accesible por las partes.

En la actualidad los proveedores Cloud ofrecen una visión muy básica de los SLAs para sus clientes. Estos acuerdos siempre se presentan como documentos de texto que especifican las obligaciones en términos de *uptime*, comprometiéndose a que el usuario disponga de un tanto por cien de uptime (medido a lo largo de un cierto periodo de tiempo). Como obligación se establece una compensación, habitualmente en forma de descuento en la

facturación, a los usuarios en caso de violación del contrato. Otros aspectos candidatos a formar parte de un acuerdo de servicio, como pueden ser los requisitos de CPU o memoria, restricciones geográficas, etc., se dejan como opciones de configuración o son ignoradas por completo por parte de los proveedores de servicio.

Para ofrecer algunas de las características definidas para una plataforma PaaS, es necesario disponer de SLAs elaborados que permitan manejar restricciones, configuraciones, información de usuario, etc. Una posible solución consistiría en la inclusión del concepto de 'SLA Digital', junto con un marco de trabajo que permita a los administradores de la plataforma la definición de sus propios términos y reglas de administración de SLA.

Un SLA Digital se puede definir como la representación binaria del documento de acuerdo de servicio, permitiendo así una versión flexible, personalizable y automáticamente procesable del contrato. Este concepto permitía integrar el SLA con la plataforma y proporcionar operaciones relacionadas con el mismo, así como definir distintos contratos para cada servicio y usuario, almacenarlos, recuperarlos y modificarlos, permitir procesos de negociación, etc. Al mismo tiempo, habilitaría a los administradores para concretar los términos y políticas del documento, ajustando el contrato a su modelo de negocio. Algunos términos básicos que se podrían incluir en un SLA se enumeran a continuación.

- Recursos requeridos por el usuario (CPU, memoria, almacenamiento).
- Restricciones sobre los servicios (por ejemplo zonas geográficas).
- Precio de los elementos, pudiendo ser especificado individualmente para cada usuario o servicio.
- Opciones de configuración de los servicios (por ejemplo reglas de elasticidad, lista de desarrolladores autorizados a administrar el servicio, número de réplicas de los datos almacenados).
- Garantías ofrecidas por el proveedor (tiempo de respuesta máximo, restricciones de seguridad, etc.).
- Términos de compensación en caso de incumplimiento del contrato.

Un SLA Digital proporciona mayores garantías de servicio debido a su integración con la plataforma y la posibilidad de procesado automático. En un entorno tradicional, el personal humano es el encargado de asegurar cumplimiento del SLA, y de cumplir con las responsabilidades y garantías definidas

en él. Sin embargo, utilizando un SLA Digital, es la propia plataforma la encargada de llevar a cabo operaciones encaminadas al cumplimiento de los contratos. De forma complementaria, la plataforma también puede realizar comprobaciones del estado de los SLA, determinar si se han violado sus términos, realizar acciones correctoras, notificar al usuario del incidente y ejecutar los procedimientos de compensación adecuados.

3.2.8. Auto-recuperación

Debido a su naturaleza distribuida y heterogénea, el correcto funcionamiento de la plataforma puede verse comprometido por una variedad de causas, por ejemplo el fallo de alguno de sus nodos. Un sistema de auto-recuperación puede apoyar a la hora de proporcionar robustez, al tiempo que permitiría ayudar a mantener el nivel de QoS establecido para la plataforma.

Auto-recuperación se puede definir como la capacidad de la plataforma para recuperarse ante la ocurrencia de situaciones adversas (por ejemplo fallo de nodos, ataques DoS), para tratar de mantener la funcionalidad de la plataforma, o mejorar la calidad de servicio ofrecido (es decir, tiempo de respuesta). Las acciones de recuperación incluyen la posibilidad de deshabilitar nodos que interfieren en el correcto funcionamiento de otros componentes, o añadir nodos y desplegar componentes necesarios para poder mantener la plataforma operativa y alcanzar su configuración óptima.

3.2.9. Migración en vivo

La migración en vivo se puede definir como el movimiento de un componente software en ejecución de una máquina física a otra, sin interrumpir el servicio ni producir un *downtime*[1] desde el punto de vista del usuario [41]. Algunas de las principales ventajas de la migración en vivo son las siguientes:

- El software puede moverse de una máquina a otra, independizando su ejecución de la máquina origen. Utilizando esta característica se permite a los administradores que intervengan una máquina sin afectar a la ejecución de la aplicación. Por ejemplo, si una máquina debe ser apagada por mantenimiento, la migración en vivo permite mover los servicios alojados en la misma a otro nodo, manteniendo su ejecución activa.

[1] El término downtime hace referencia al periodo del tiempo durante el cual un sistema software no puede proporcionar o realizar su tarea primaria, habitualmente debido a fallos o paradas de mantenimiento programadas.

- Los servicios y aplicaciones se pueden migrar entre recursos, aplicando algoritmos, con el objetivo de equilibrar la carga. De esta forma, si una máquina está sobrecargada y otra tiene capacidad libre, sería posible mover servicios entre ambas optimiza la utilización de los recursos y mejora el rendimiento. Esto podría implicar incluso a los propios servicios que conforman la infraestructura.

- La migración en vivo se realiza sin interrumpir el servicio y, por tanto, los usuarios pueden continuar utilizando el servicio sin percibir que su aplicación ha sido trasladada a otra máquina física.

- Usando interoperabilidad y migración en vivo, los desarrolladores pueden, potencialmente, cambiar de proveedor de una forma casi automática.

- El auto escalado (de aumento y de disminución) resulta más sencillo y eficiente ya que se puede apoyar en este tipo de herramientas. Por un lado, al escalar hacia arriba, la carga se puede redistribuir entre los nodos, mientras que cuando se escala hacia abajo, los servicios de las máquinas que se van a parar pueden ser movidos a recursos físicos que van a permanecer en funcionamiento.

La migración en vivo ha sido considerada como una característica interesante también para las soluciones Cloud basadas en IaaS, debido a las numerosas ventajas que ofrece. En este sentido, se han realizado investigaciones orientadas a habilitar y optimizar la migración de máquinas virtuales para soluciones IaaS [42].

La migración en vivo se puede llevar a cabo utilizando un protocolo de migración. Este tipo de protocolos consisten habitualmente de tres pasos básicos: despliegue de una nueva instancia, redirección de los clientes de la instancia antigua a la nueva y repliegue de la instancia antigua.

La migración en vivo que implica transferencia de estado es dependiente de la aplicación, y por tanto necesita de la participación activa de los desarrolladores en el proceso. En este sentido, la *migración en vivo con estado* puede llegar a lograrse proporcionando herramientas adicionales a los desarrolladores. Utilizando estas herramientas, los desarrolladores pueden implementar las funciones de migración de datos necesarias por la aplicación, y la plataforma podría llevar a cabo las llamadas oportunas cada vez que se produzca este tipo de migración.

3.2.10. Versionado

Uno de los objetivos de las soluciones PaaS debe ser el de integrar el proceso de desarrollo de software en la plataforma. La idea subyacente es, esencialmente, que el proceso de desarrollo sea cloud-aware, para permitir desarrollar aplicaciones Cloud y no simplemente aplicaciones que se ejecutan en la nube ya que en ese caso probablemente bastaría con los entornos IaaS tal como están siendo considerados en la actualidad. El resultado de esta integración está encaminado a permitir a los desarrolladores adaptar el ciclo de desarrollo a la nube, creando aplicaciones que tienen como *runtime* la plataforma PaaS.

Uno de los pasos importantes de esta adaptación debe pasar por considerar el versionado de los desarrollos, ya que esto va a permitir mantener un ciclo de desarrollo incremental, como pieza clave en la ingeniería del software. De esta forma, utilizando la característica de versionado, los desarrolladores van a poder mantener servicios de larga duración ejecutándose en la nube, al tiempo que estarán habilitados a añadir nuevas funcionalidades incrementalmente, y testear nuevas versiones de servicios en el entorno de ejecución antes de ser publicadas.

Una de las principales aplicaciones del versionado es la de permitir desplegar versiones estables de un software en la nube y, concurrentemente, desplegar versiones para testeo. Posteriormente, cuando las versiones en desarrollo han sido corregidas y se pueden considerar estables, las instancias del servicio desplegadas pueden actualizarse al vuelo a la nueva versión, proporcionando a los usuarios las nuevas funcionalidades de forma transparente, siempre que el desarrollador tenga en cuenta los aspectos de compatibilidad.

3.2.11. Catálogo

En una plataforma Cloud, se dispone de una gran cantidad de información distribuida en los componentes que conforman la misma. Alguna de esta información incluye, por ejemplo, los registros de los usuarios, los registros de los recursos, la información generada por la monitorización, etc. Esta información se genera en los distintos módulos que componen la plataforma, y su almacenamiento y gestión proporciona ventajas tanto a los administradores como a los usuarios de la plataforma PaaS.

Por un lado, para su correcto funcionamiento, la plataforma necesita de cierta información, como los usuarios que pueden acceder a ella o los recursos que se encuentran disponibles. Por otro lado, los administradores requieren ciertos datos para conocer el estado de la plataforma y realizar estadísticas

y análisis, con el objetivo de mejorar el propio despliegue de la solución. Utilizando información sobre la monitorización del sistema los administradores pueden conocer problemas en el funcionamiento de la plataforma, determinar cuellos de botella, definir patrones de comportamiento u obtener otras conclusiones que facilitan la administración de la misma.

En otro plano los desarrolladores necesitan la información relativa al funcionamiento de sus servicios para poder conocer su estado y mejorar su comportamiento. Y finalmente los consumidores de los servicios precisan de sistemas de descubrimiento para buscar servicios alojados en la plataforma.

Para poder satisfacer las necesidades de los distintos usuarios de la plataforma en este aspecto, es necesaria la utilización de distintos sistemas de información que recojan todas las características mencionadas. Sin embargo, de entre las posibles opciones para la implementación de un sistema de información, podemos seguir una aproximación integral, en forma de catálogo de la plataforma.

Podemos considerar que un catálogo será el sistema donde almacenar y gestionar de forma conjunta la información generada en la plataforma, organizarla y dar acceso a la misma. De esta forma, los posibles subsistemas de información (monitorización, gestión de servicios, etc.) podrán disponer de la información conjunta para correlacionarla y proporcionar datos más elaborados. La utilización de un catálogo distribuido, sobre otras alternativas para el almacenamiento de información (como que cada componente aloje sus propios datos) proporciona una serie de ventajas, tal y como se enumera a continuación.

- Ubicuidad: A pesar de la naturaleza distribuida del catálogo, ligado a la propia distribución de los componentes de la plataforma, este permite acceder a toda la información alojada a través de distintos puntos de acceso al mismo, con el objetivo de que los usuarios puedan obtener la completitud de los datos sin la necesidad de consultar los distintos componentes de forma individualizada.

- Interfaces: La utilización de un catálogo como sistema único permite ofrecer un interfaz bien definido para el acceso a toda la información alojada en el sistema, por dispar que esta sea, en lugar de disponer de distintos interfaces a distintos sistemas de información.

- Desacoplamiento: El hecho de disponer de una lógica de almacenamiento y acceso a la información propia del catálogo, en un módulo diferenciado del resto de componentes, permite desacoplar dicha lógica de la del resto de componentes del sistema. De esta forma, modificando

el componente de catálogo se puede modificar el tipo de información almacenada, la forma de almacenamiento, los interfaces de acceso, etc. sin necesidad de implicar al resto sistemas.

3.2.12. Interoperabilidad

En la actualidad, cada solución PaaS comercial es propiedad de un solo proveedor, y por razones comerciales no se genera un interés en permitir la interoperabilidad entre los distintos proveedores de servicio. Por otro lado, la falta de estándares en el Cloud complica las posibles interacciones entre distintas nubes, puesto que cada plataforma define sus propios interfaces, operaciones y protocolos. Los componentes de interoperabilidad están encaminados a permitir la interacción entre distintas nubes. De esta forma, haciendo uso de dichos componentes, una nube saturada podría alquilar recursos de otra nube con capacidad libre, pudiendo así hacer frente a un pico carga y permitiéndole cumplir con los SLA de usuario. Dichos componentes de interoperabilidad también permite la creación de entornos colaborativos donde nubes de distintas entidades se federan en una plataforma común, abarcando distintos dominios administrativos, de la misma forma en la cual los sistemas Grid trabajan colaborativamente en proyectos tales como EGEE [43].

La interacción entre nubes se puede entender como la cesión de recursos entre dos nubes (potencialmente, a cambio de una contraprestación). Sin embargo, la interacción entre nubes no solo implica el modo en que dos nubes intercambian recursos o elementos de ejecución, sino que dada su propia naturaleza, debe incluir procesos de negociación de recursos [44], donde se involucran la nube consumidora y la nube proveedora. Un proceso de negociación se compone de tres elementos, los cuales se detallan a continuación.

- Proceso de negociación: Este punto establece cómo dos nubes intercambian mensajes y logran alcanzar un acuerdo mediante la solicitud de recursos, evaluación de ofertas, planteamiento de contraofertas, etc. En el campo de la IA se han definidos algoritmos de negociación para agentes que pueden encajar en los requisitos del proceso de interoperabilidad, tal y como se muestra en la Figura 3.1.

- Lenguaje de comunicación: Este punto define cómo las nubes representan e intercambian información. Este lenguaje incluye la definición de los recursos solicitados, su precio, términos de uso, etc.

- Procedimiento de intercambio: Este punto define cómo los recursos

alquilados son incorporados a la nube consumidora. Un aspecto esencial de este procedimiento es que debe asegurar que la nube consumidora puede hacer uso de los recursos sin que la nube proveedora pierda el control o la posesión sobre ellos puesto que ésta sigue siendo la propietaria.

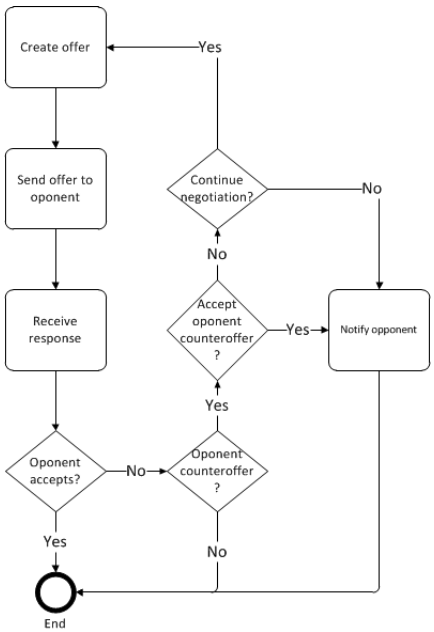

Figura 3.1: *Algoritmo de un proceso de negociación. El consumir empieza por Create offer, mientras que el proveedor empieza por Receive response.*

Dada la situación actual de diversidad de nubes y falta de estándares, se puede complementar la funcionalidad de este componente mediante un sistema de plug-ins para permitir que una plataforma pueda alquilar recursos disponibles en soluciones comerciales. El plug-in no sólo debería actuar como intermediario de negociación entre las nubes, sino que necesitaría incluir mecanismos para añadir los recursos de las soluciones comerciales a la nube privada.

3.3. Arquitectura

Una vez especificadas las características con que debe contar la plataforma y descrita la visión general de la misma, nos centramos en el diseño de

una arquitectura que permitirá dar soporte a las mismas.

Para favorecer la modificación y adaptación de la herramienta, así como su correcto funcionamiento, se ha planteado una arquitectura modular, distribuida y débilmente acoplada. Así, la arquitectura se divide en bloques lógicos heterogéneos que se encuentran físicamente distribuidos, y donde cada uno de ellos implementa una funcionalidad concreta. Siguiendo esta aproximación, obtendremos las siguientes ventajas:

- La plataforma se encuentra físicamente distribuida, y con algunos componentes replicados, aumentando así la robustez del sistema.
- El diseño, implementación y depuración resulta más sencillo, al separar la plataforma en componentes lógicos y desacoplar interfaz e implementación.
- Cada módulo pueda ser modificado o sustituido sin necesariamente afectar al correcto funcionamiento de la plataforma.
- Algunas de las características necesarias para la plataforma, como el auto escalado, se benefician de este esquema, puesto que es posible desplegar nuevas instancias de módulos concretos, en lugar de la plataforma al completo.

Se han definido un conjunto de módulos y capas de abstracción que proporcionan toda la funcionalidad descrita en el apartado de especificación. En la Figura 3.2 se muestra la arquitectura general de la plataforma.

En esta figura, las cajas verticales representan unas capas de abstracción, que son comunes a toda la plataforma, mientras que las cajas horizontales representan módulos. En nuestro caso hemos considerado que una capa de abstracción es, de forma resumida, un interfaz común usado por los componentes, independientemente de la implementación subyacente del sistema concreto. De acuerdo a la arquitectura, las capas de abstracción son el almacenamiento y la seguridad y, mediante este mecanismo, obtenemos la posibilidad de disponer de distintos sistemas de almacenamiento y esquemas de seguridad, habilitando a su vez que estos sean extensibles y mejorables. Por su parte los módulos representan elementos funcionales del sistema.

En los siguientes apartados se detalla el diseño de cada uno de los módulos, explicando la funcionalidad que desempeñan dentro del contexto de la arquitectura, así como las dos capas de abstracción contempladas.

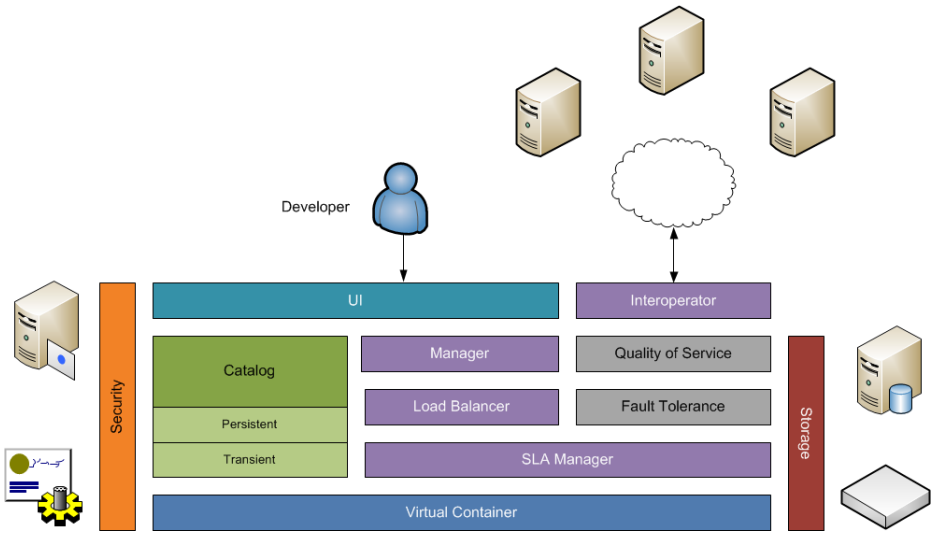

Figura 3.2: *Arquitectura del sistema. Cada color denota distintos bloques conceptuales del sistema.*

3.4. Módulos

Cada módulo implementa una funcionalidad específica distinta, y se pueden instanciar individualmente. Las instancias se agrupan formando un clúster global del despliegue, en el cual todos los componentes se conocen y pueden comunicarse entre sí mediante paso de mensajes. Una instancia de la plataforma completa (o nube privada) estará compuesta por, al menos, una instancia de cada componente.

Puesto que esta arquitectura está orientada a la oferta de servicios, resulta natural que los propios módulos del sistema estén implementados como tales. De esta forma, cada uno de los componentes se empaqueta en un servicio que ofrece su funcionalidad al exterior mediante un interfaz. Estos servicios son los que denominamos *core services* puesto que son los que componen el núcleo de la plataforma.

Para poder ejecutar servicios, tanto los *core services* como los de los usuarios de la plataforma, es necesario un *runtime* donde alojarlos. Dicho *runtime* actúa como middleware entre la máquina física y el entorno virtual de la solución, permitiendo desplegar una plataforma en una infraestructura física.

Este *runtime* se ha incluido en el componente *Virtual Container*, que se compone de un contenedor de servicios tradicional (como Apache Axis, o

Metro), con una serie de funcionalidades extra, que tiene que ser puesto en funcionamiento junto con un servicio que implementa el interfaz al exterior del componente *Virtual Container*. El objetivo es que el funcionamiento del *Virtual Container* sea equivalente al de un hipervisor en el caso de las máquinas virtuales, actuando de middleware entre la máquina física y el software virtualizado que se ejecuta en ella.

3.4.1. Capas de abstracción

Las capas de abstracción representan funcionalidades transversales a la plataforma, que son utilizadas por múltiples componentes de la misma, y que tienen un cometido concreto. Estas funcionalidades se ofrecen como Interfaces que incluyen las operaciones básicas designadas para cada caso, independientemente de la implementación subyacente. De esta forma, estas Interfaces se diseñan como APIs, cuyas implementaciones son las que se distribuyen de forma efectiva junto con los *core services* y soportan el funcionamiento de la plataforma.

Para Cloud ComPaaS hemos definido dos capas de abstracción básicas: el almacenamiento y la seguridad. Estas capas proporcionaran una serie de funciones, que permiten independizar el interfaz y la funcionalidad de la implementación efectiva.

Storage

El objetivo de esta capa de abstracción es el de ofrecer un API para poder acceder a un sistema de almacenamiento Cloud. La implementación subyacente la determina las características de la infraestructura que la aloja y, en definitiva, los administradores de la plataforma. Esta capa puede tener múltiples implementaciones que pueden ser utilizadas de forma conjunta.

El interfaz de almacenamiento ofrece operaciones estándar, como son lectura y escritura de ficheros, y da acceso a un sistema de ficheros virtual privado para cada usuario. Las implementaciones de este API deben cumplir los principios Cloud como son ubiquidad, replicación, control de acceso, etc., aunque los detalles de las características que finalmente aporta el sistema de almacenamiento al despliegue quedan en manos de los administradores mediante la plataforma de almacenamiento a la que decidan dar soporte.

De esta forma, el almacenamiento se puede implementar como conexiones a Amazon S3, a un sistema GridFTP, utilizar un sistema de ficheros en red, o simplemente utilizando el disco local. Algunas soluciones, como esta última, no cumplirían con todos los principios Cloud mencionados anteriormente,

pero por un lado resultarían inmediatas de implementar y, por otro, pueden ofrecer un mejor rendimiento de acceso a los ficheros.

Security

La capa de seguridad ofrece un API para un marco de seguridad que incluye una serie de operaciones básicas, como son autentificar usuario u obtener datos del mismo. El control de acceso en la plataforma se realiza en base a roles de usuario, contando con tres grupos básicos definidos: administrador, desarrollador y consumidor. Adicionalmente, se considera un rol especial Cloud, orientado a la interoperabilidad, y que representa la identidad de la plataforma, y para ello debe contar con unas credenciales propias.

Desde el punto de vista de la plataforma, la seguridad consiste en determinar qué usuarios están autorizados para realizar qué acciones y, son los roles básicos los que determinan los derechos de acceso, en una jerarquía decreciente inclusiva (es decir, los roles superiores incluyen a los inferiores). De forma genérica, los derechos de cada uno de los roles se define de la siguiente forma.

- Administrador: Es el rol con la capacidad de interactuar directamente con los componentes del sistema a través de operaciones no públicas. La misión de este rol es la de administrar usuarios, servicios, componentes, modificar la configuración del sistema, etc.
- Desarrollador: Es el rol con la capacidad de registrar y desplegar servicios en la plataforma. Entre otra información, cada desarrollador tiene asociada un SLA que especifica, por ejemplo, los recursos a los que tiene acceso. A partir de este SLA el desarrollador puede establecer restricciones en cuanto a los servicios que despliega en la plataforma, como requisitos mínimos de memoria, CPU, número de instancias, etc.
- Consumidor: Es el rol más básico, pues solamente proporciona la capacidad de utilizar servicios alojados en la plataforma y realiza consultas públicas al catálogo.
- Cloud: El rol de Cloud está reservado para representar a aquellas otras plataformas que se han registrado como usuarias del sistema. Sus capacidades son similares a las del rol de desarrollador, aunque en lugar de permitir la solicitud del registro de servicios, realiza solicitudes de cesión de recursos. Los SLA asociadas al rol Cloud son más complejas que las de desarrollador, puesto que deben especificar aspectos como el coste de los recursos o las restricciones sobre las cesiones.

- Credenciales de la plataforma: La identidad implícita de la plataforma se usa como elemento de seguridad, si bien no se explicita como un usuario registrado en el sistema. Así, la plataforma en sí dispone de unas credenciales propias que la identifican y que sirven para determinar qué elementos forman parte del sistema y cuáles no. Estas credenciales se utilizan para autentificar a los componentes que se unen a la plataforma, y poder establecer un nivel de seguridad en las comunicaciones que se llevan a cabo entre dichos componentes a nivel interno.

3.4.2. Virtual Containers

Los contenedores virtuales son el componente básico de la plataforma, pues es el elemento que actúa de middleware entre la infraestructura física y la plataforma virtual distribuida. Este es el componente encargado de virtualizar el entorno de ejecución y aloja de forma efectiva los servicios virtuales. Su funcionamiento básico se ilustra en la Figura 3.3.

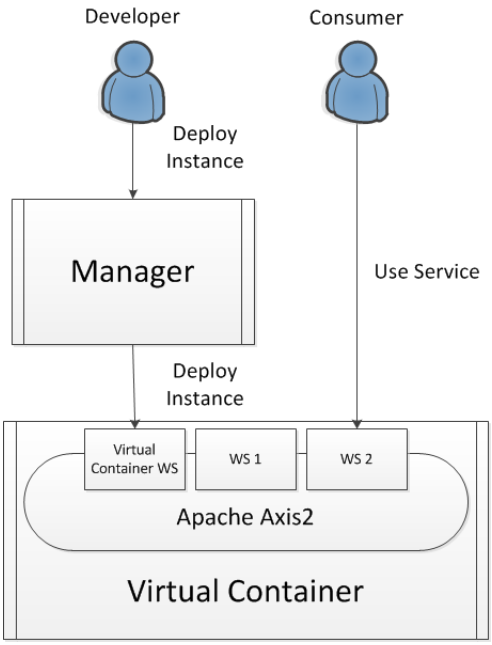

Figura 3.3: *Los desarrolladores solicitan desplegar instancias al módulo de Manager, mientras este redirige las peticiones al Virtual Container seleccionado. El consumo de servicios se realiza de forma directa.*

Un *Virtual Container* es un contenedor de servicios web (tales como Apache Axis2 o Apache CXF) que ha sido virtualizado para ofrecer una serie de características propias de los entornos virtuales. Las características que debe ofrecer un *Virtual Container* son:

- Independencia del hardware y de la plataforma: El entorno ofrece una visión homogénea de la plataforma con independencia del hardware o sistema operativo subyacente.
- Aislamiento: El entorno que utiliza un usuario está aislado de otros entorno, evitando que distintos usuarios puedan interactuar de forma directa, o que un usuario pueda interferir en los recursos y servicios de otro.
- Particionado de recursos: El entorno puede particionar los recursos (CPU, memoria, etc.) de forma discreta. Este particionado permite que diferentes instanciaciones del entorno puedan tener distinta cantidad de recursos, potencialmente diferente al de la máquina física.
- Operaciones de administración: El entorno ofrece un conjunto de funciones que permiten administrar los servicios que en ella se van a alojar.
- Monitorización: El entorno monitoriza su propio estado y las acciones que ocurren en su interior. Como resultado de esta monitorización se genera información que es almacenada para su posterior consulta y análisis.

Para que un contenedor de servicios web pueda ofrecer estas características, es necesario modificarlo o añadirle componentes que posean la funcionalidad deseada, mientras que en algunos casos, la propia naturaleza del contenedor cumple con alguno de estos requisitos.

Como parte del diseño del módulo *Virtual Container*, se definieron las adaptaciones necesarias para que un contenedor de servicios web pueda contener todas estas características.

- Independencia del hardware y de la plataforma: Los contenedores de servicios web actúan como *runtime* de los servicios. Por tanto, cualquier contenedor que se utilice independiza la ejecución de los servicios web del hardware y sistema operativo subyacente.
- Aislamiento: Puesto que el lenguaje de programación soportado por Cloud ComPaaS es Java, la propia máquina virtual de Java proporciona un entorno *sandbox* que aísla los servicios ejecutados en distintos

contenedores, aun cuando dichos contenedores se ejecutan en la misma máquina física.

- Particionado de recursos: La máquina virtual de Java permite definir el tamaño de memoria del entorno de ejecución. En cuanto al resto de recursos, como CPU o disco duro, es necesario acudir a herramientas externas como limitadores de CPU o sistemas de ficheros virtuales, que permiten dividir los recursos de la máquina física en porciones de menor tamaño.

- Operaciones de administración: La forma más sencilla de incorporar funciones de administración a un contenedor sin modificar su código fuente es introducirlas como un servicio. Todas las funciones de administración que son utilizadas a nivel interno por el contenedor se implementan en un servicio *Virtual Container*. Este servicio se aloja en el contenedor de servicios, y mientras esté activo, este ofrecerá al exterior las funcionalidades del *Virtual Container*. Una ventaja adicional de esta aproximación es que mantiene la coherencia con el resto de la plataforma, pues cada componente se implementa como un servicio web.

- Monitorización: La monitorización del *Virtual Container* se contempla a dos niveles: de forma reactiva, monitorizando eventos producidos por acciones externas, como por ejemplo una solicitud de despliegue de instancia, y de forma proactiva, monitorizando el estado del componente, como por ejemplo la cantidad de CPU consumida en un instante determinado.

La monitorización reactiva se lleva a cabo desde el servicio web *Virtual Container*, pues es el componente que recibe todas las peticiones externas. La monitorización proactiva se puede realizar mediante un proceso en segundo plano que periódicamente realiza medidas sobre ciertos recursos, como son la CPU, la memoria, el número de servicios desplegados, etc. La monitorización de aspectos más detallados, como el número de solicitudes que recibe un servicio, requiere de la modificación del código fuente del contenedor de servicios, pues es este el que procesa esta información a nivel interno.

Un *Virtual Container* es, por tanto, un contenedor de servicios web que cumple con todos los requisitos especificados o al cual se le han llevado a cabo las modificaciones mencionadas para que los cumpla. Las funciones de administración que ofrece son las siguientes.

- Desplegar instancia: Despliega la instancia de un servicio en el contenedor virtual. Para ello recupera el paquete binario del servicio (mediante la utilización del catálogo) y lo pone en funcionamiento, registrando la información de la instancia en el catálogo.

- Replegar instancia: Finaliza la ejecución de una instancia, eliminando la información asociada del catálogo y eliminando el paquete binario del servicio.

- Actualizar instancia: Repliega la versión anterior de un servicio y despliega la versión nueva, haciendo uso de la información proporcionada.

- Migrar instancia: Esta función se invoca en los contenedores origen de la migración de un servicio. En caso de que el servicio que se va a migrar contenga estado, y los desarrolladores hayan implementado un interfaz de migración proporcionado por la plataforma, el contenedor virtual invoca a un método que se ocupa de realizar la migración, que será el encargado de habilitar al servicio para llevar a cabo la migración de su estado. Una vez finalizado, el contenedor repliega la instancia.

Los *Virtual Container* son el recurso que se pone a disposición de los desarrolladores, de forma análoga a las máquinas virtuales en las nubes IaaS. Los desarrolladores inscritos en la plataforma tienen la capacidad de solicitar el despliegue de nuevos *Virtual Container*. La plataforma, atendiendo a los requisitos del desarrollador, las capacidades y limitaciones definidas por su SLA de usuario y los recursos físicos disponibles, despliega un nuevo contenedor virtual y lo asocia al usuario. A partir de ese momento, el contenedor será un candidato a albergar instancias de servicios del desarrollador, siempre que tenga capacidad suficiente para albergarlos.

3.4.3. Catalog

El catálogo es el componente encargado de almacenar y gestionar información acerca de la plataforma, desde los propios datos de los servicios y las instancias registradas, hasta la información de monitorización. Toda esta información es obtenida de forma pasiva por el catálogo, siguiendo una estrategia *push* por parte de los componentes, mediante la cual se reciben informes del estado de cada uno de ellos, bien de forma periódica, o bien como respuesta a ciertos eventos. El catálogo ofrece unas funciones básicas para la administración de la información que se almacena en el mismo, entre las que se incluyen las siguientes:

- Inserción: Realiza la inserción de un conjunto de registros en el sistema de información subyacente. El catálogo no realiza comprobaciones sobre la corrección del formato de los registros, y en su lugar delega dichas verificaciones en la base de datos, gestionando las excepciones debido a datos mal formateados.

- Borrado: Elimina los registros de la base de datos que coinciden con la cláusula proporcionada.

- Consulta: Devuelve los registros almacenados en la base de datos que coinciden las condiciones indicadas.

- Actualización: Actualiza los registros de la base de datos que coinciden con unas restricciones, haciendo que incluyan la nueva información suministrada.

- Info: Devuelve el nombre y tipo de datos de los campos que componente el esquema de la base de datos subyacente.

A la hora de considerar el tipo de información que se almacena en el catálogo, podemos diferenciarlas en dos grupos en base a su naturaleza: información persistente e información transitoria. Las diferencias entre ambas son las siguientes:

- Información Persistente: Incluye datos estáticos sobre la plataforma como información sobre servicios, instancias, SLA, usuarios, etc. Esta información conforma los registros de la plataforma, y es necesario almacenarla para su posterior utilización. Habitualmente esta información se genera como consecuencia de la interacción de los usuarios.

- Información transitoria: Ofrece una instantánea del estado de la plataforma, con información puntual sobre distintas características como el número de servicios, usuarios, instancias, módulos, etc. También incluye información sobre los recursos disponibles o en uso. Esta información cambian constantemente y pierde validez con el transcurso del tiempo (por ejemplo la carga de un *Virtual Container*), y por tanto debe ser actualizada de forma periódica. Habitualmente esta información la generan los propios componentes de la plataforma.

La relevancia de la diferenciación entre ambos tipos de información radica en que el soporte que se debe proporcionar a las mismas no requiere de las mismas condiciones.

Un catálogo que almacena información persistente posee el comportamiento de una base de datos replicada. Estos catálogos almacenan bases de datos que deben mantener su estado entre ejecuciones, y por tanto contienen toda la información replicada en cada nodo. Su lógica de administración determina que las consultas a la base de datos pueden ser servidas por una réplica, mientras que las modificaciones deben ser distribuidas entre todas ellas. Se debe definir un protocolo para la recuperación del estado de la base de datos en el arranque del catálogo.

Un catálogo que almacena información transitoria posee el comportamiento de una caché distribuida. Estos catálogos almacenan bases de datos que no deben ser mantenidas entre ejecuciones, y por tanto no es necesario que todos los datos se almacenen en todos los nodos. Su lógica de administración determina que las operaciones de consulta se deben distribuir a todas las réplicas, mientras que las modificaciones pueden ser atendidas por una sola. Se debe definir un protocolo que equilibre la información almacenada entre los nodos, y la replique en un número de elementos k, con k menor que el número de nodos, para aumentar la robustez y mejorar el rendimiento del sistema.

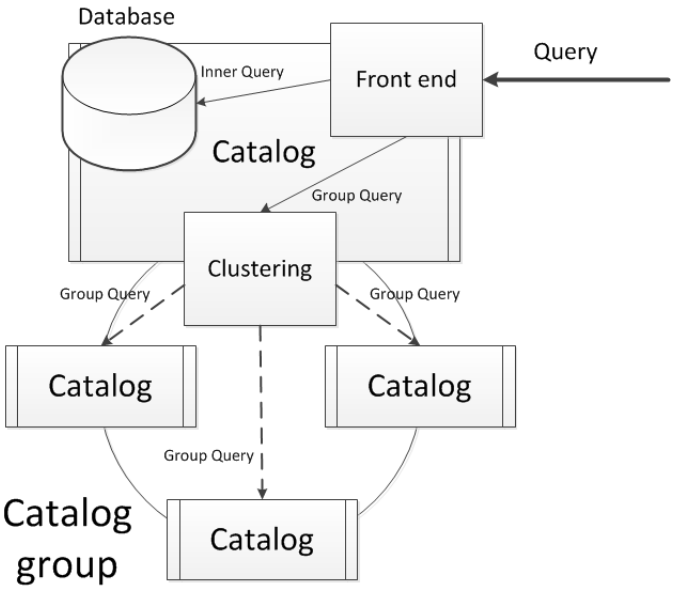

Figura 3.4: *Funcionamiento básico de un grupo de catálogos. El Front end recibe peticiones externas, que son atendidas por un mismo nodo o redirigidas al resto de nodos a través del clustering.*

El módulo catálogo se ha separado en tres bloques lógicos destinados a aislar las distintas funcionalidades del componente, aumentar su modularidad, proporcionar mayor flexibilidad y facilitar la implementación. Estos bloques son clustering, base de datos y *front end*. Los bloques de bases de datos y front-end tienen una variante para cada uno de los catálogos, persistente y transitorio, mientras que el bloque de clustering es común a ambos. Su organización básica se representa en la Figura 3.4.

- El bloque de clustering se encarga de administrar la relación entre los nodos del catálogo. La organización de los catálogos se ha planteado como sistemas P2P distribuidos, inspirado por modelos P2P de sistemas Grid [45]. La información se aloja en los nodos distribuidos, que están asociados entre sí en grupos y que a su vez pertenecen a una plataforma. Esta organización aumenta la robustez de la plataforma, pues no dispone de punto de fallo único, y además permite realizar equilibrado de cargas. Como contrapartida, debido a que la información se encuentra distribuida, ciertas interacciones con el catálogo deben ser redirigidas a todo el grupo siguiendo los esquemas P2P.

- El bloque de bases de datos define el tipo de información que se va a almacenar, así como la forma en la que se va a realizar dicho almacenamiento. Este bloque define el motor de bases de datos a utilizar, las operaciones de almacenamiento disponibles y el esquema de bases de datos incluyendo las tablas, campos, tipos de datos, etc. Las variantes persistente y transitoria se diferencian en que la base de datos subyacente para el catálogo persistente es una base de datos habitual, mientras que la del catálogo transitorio se almacena solamente en memoria, sin generar ficheros en la máquina ni mantener los datos entre ejecuciones.

- El bloque de *front end* se encarga de implementar la lógica de los catálogos y definir el interfaz de los mismos. Esta lógica define el comportamiento de los catálogos frente a las interacciones de los usuarios, así como su comportamiento en el arranque y la finalización de la ejecución. Este bloque es el encargado de proporcionar las funciones básicas de administración de información enumeradas anteriormente. De acuerdo a los dos tipos de catálogo que estamos manejando, nos encontraremos con dos front-ends diferenciados. Por un lado, el *front end* persistente implementa la lógica básica de un catálogo utilizando una base de datos persistente. En un catálogo persistente, cada nodo debe poseer una réplica de la base de datos entera, para poder mantener la

coherencia entre ellas. Debido a esta característica, las operaciones de consulta pueden ser servidas por un solo nodo, mientras que las operaciones de inserción, borrado y modificación deben ser distribuidas entre todos los nodos. Por otro lado, nos encontramos con el *front end* transitorio, que implementa la lógica básica de un catálogo transitorio. En esta organización, la información no se encuentra necesariamente replicada en cada uno de los nodos del sistema. Por este motivo, y debido a la organización distribuida, las operaciones de consulta deben ser trasladadas a todos los nodos del grupo para obtener la respuesta completa a la misma, mientras que las operaciones de inserción, borrado y modificación pueden ser atendidas por un solo nodo.

Estos bloques definen un catálogo abstracto el cual posee la funcionalidad básica, pero carece de información u operaciones concretas. Para lograr un catálogo funcional, es necesario implementar el diseño de una base de datos y el interfaz de un *front end* que genere una concreción del catálogo. Una concreción del catálogo es un servicio web que implementa un catálogo del sistema, con una información y operaciones determinadas.

3.4.4. Manager

El *Manager* es el modulo encargado de gestionar todas las operaciones de administración de la plataforma. Su funcionalidad consiste en ofrecer a los usuarios funciones relacionadas con servicios, instancias, SLAs, usuarios, etc. Estas funciones se enumeran a continuación.

- Alta de servicio: Dar de alta un servicio es la acción de registrar el paquete y los metadatos de un servicio en la plataforma. Esto se traduce en una interacción con el sistema de información de la plataforma (en nuestro caso, el catálogo), donde se debe registrar el propio servicio y la información asociada a este. El *Manager* registra, entre otros aspectos, el paquete binario, lo relaciona con el usuario que ha solicitado dicha operación, los metadatos del servicio y el SLA digital asociada al mismo. Adicionalmente realiza comprobaciones funcionales y de integridad, como que no existe un servicio con el mismo nombre, o que el usuario posee derechos para registrar una versión del servicio indicado.

- Baja de servicio: Dar de baja un servicio consiste en eliminar el servicio de la plataforma. Esto implica tanto la eliminación del registro (como funcionalidad recíproca al alta del servicio), como el repliegue de todas las instancias derivadas de este. El módulo realiza comprobaciones

funcionales, entre las que se encuentra la verificación de la autorización del usuario que solicita la operación para dar de baja el servicio.

- Actualización de servicio: Actualizar un servicio consiste en sustituir todas las instancias de una versión determinada de un servicio por instancias de otras versiones. El procedimiento de actualización consiste en replegar las instancias en ejecución de una determinada versión, desplegando en su lugar la versión actualizada del mismo. Este proceso debe ser transparente para el usuario, que accede a la versión nueva del servicio de forma convencional a la hora de realizar nuevas peticiones a la instancia, con las lógicas restricciones de compatibilidad determinadas por el creador del servicio. Un algoritmo de actualización debe tratar de minimizar el tiempo de no disponibilidad del servicio, al actualizar las instancias secuencialmente en lugar de en paralelo. Por otro lado, en el catálogo de la plataforma se debe reflejar la actualización de las versiones.

- Migración de servicio: La migración de servicios, como se ha comentado en apartados anteriores, consiste en replegar una instancia de un servicio en una localización y desplegarla en otra distinta, de tal forma que pueda continuar funcionando con normalidad en la nueva ubicación. En este caso el *Manager* debe realizar comprobaciones de integridad y funcionamiento en las que no sólo se ha de verificar la autorización del usuario para solicitar la migración, sino que debe asegurarse que dicha migración no supongo una violación de el SLA asociada a la instancia.

- Despliegue de instancia: Desplegar una instancia asociada a un servicio consiste en poner en ejecución una instancia de un servicio registrado en el sistema de información de la plataforma. El *Manager* debe comprobar aspectos como los permisos del usuario para registrar instancias y el acceso al servicio solicitado, así como que dicho registro respete el SLA del mismo (por ejemplo el usuario ha alcanzado el máximo de instancias permitidas). El despliegue de las instancias quedará reflejado en el catálogo de la plataforma, como información disponible tanto para su propio funcionamiento, como para que pueda ser consultado por parte de los usuarios.

- Repliegue de instancia: Replegar una instancia consiste esencialmente en parar la ejecución de una instancia de un servicio. De forma lógica se deben realizar las comprobaciones funcionales y de integridad asociadas a este proceso, como son la verificación de los permisos y SLAs. El

resultado de esta operación se refleja en el catálogo de la plataforma eliminando el registro de la instancia.

- Alta usuario: Se da de alta un nuevo usuario en el sistema.
- Baja usuario: Se da de baja un usuario del sistema.
- Modificar datos de usuario: Se modifican los datos de un usuario del sistema.

Las funciones del *Manager* son públicas, aunque cada una está orientada a un tipo de usuario distinto. Por ejemplo, los usuarios con rol desarrollador tienen permisos para registrar servicios, mientras que no tienen permisos para dar de alta usuarios, operación reservada al rol administrador. Todas las operaciones se rigen por un esquema genérico que se detalla a continuación y se muestra en la Figura 3.5.

1. Verificación de la autorización: En primer lugar se comprueba que el usuario tiene los permisos adecuados para solicitar la operación indicada.
2. Consulta del catálogo: Se realiza una consulta al catálogo para obtener la información necesaria para llevar a cabo la operación, por ejemplo los metadatos de un servicio.
3. Verificación de SLAs: Se comprueba que los SLAs implicadas (usuario, servicio, instancia, etc.) en la operación son correctas y permiten la aplicación de la operación.
4. Realización de la operación: Se realiza la operación solicitada.
5. Actualización del catálogo: Se actualiza el catálogo con la información que ha sido añadida, eliminada o modificada.

3.4.5. Quality of Service

El módulo de Quality of Service es el encargado de realizar las operaciones autónomas de la plataforma. Este módulo no ofrece ninguna función ni pública ni privada al exterior, si no que se ejecuta de forma continua realizando tareas de monitorización e interactuando de forma proactiva con el resto de componentes, tal y como se ilustra en la Figura 3.6. Los objetos que monitoriza el módulo son los siguientes:

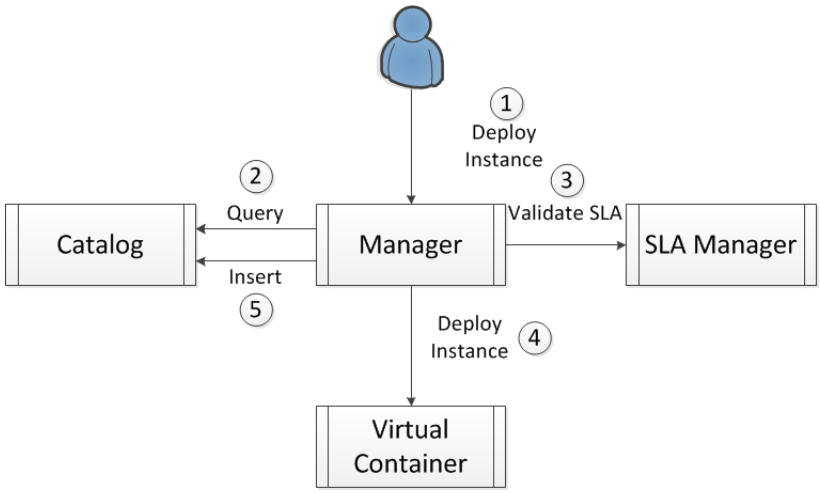

Figura 3.5: *Interacción de los componentes implicados en el despliegue de una instancia en la plataforma.*

- SLAs: El módulo comprueba de forma periódica los SLA de todos los elementos de la plataforma verificando que estos se cumplen en el instante actual. Para ello recupera los SLA almacenados en el catálogo y hace uso del *SLA Manager* para comprobar la validez de los mismos.
- Indicadores de calidad de servicio: Estos indicadores son unos parámetros orientados a medir valores que aportan información significativa sobra la calidad del servicio que ofrece la plataforma. Los parámetros suelen ser valores obtenidos de las tareas de monitorización del sistema (por ejemplo carga de los elementos). Estos indicadores se confrontan a unas reglas de elasticidad, que indican si esos valores son considerados correctos para la configuración actual de la plataforma.

Con la información obtenida de estas comprobaciones, el módulo realiza operaciones orientadas a mejorar la calidad de servicio.

- Migración en vivo: La migración dinámica de elementos en la plataforma tiene como objetivo equilibrar la carga, reducir latencias, mejorar el rendimiento, etc. Para realizar estas acciones, el módulo se comunica con el *Virtual Container* que aloja la instancia objeto de migración.
- Auto escalado: El despliegue o repliegue de componentes de la plataforma tiene como objetivo principal aumentar la capacidad de la plataforma (si se aumenta) u optimizar la utilización de los recursos

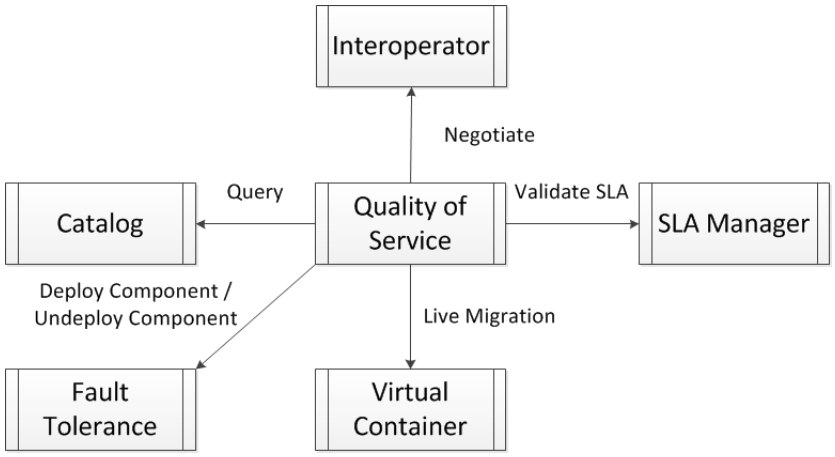

Figura 3.6: *Interacción del módulo Quality of Service con el resto de la plataforma.*

(si se disminuye). Unas reglas de elasticidad indican cuando es conveniente aplicar estas técnicas para mejorar los indicadores de calidad de servicio. Para realizar estas acciones el módulo Quality of Service interactúa con el de *Fault Tolerance*.

- Acciones correctoras: Bajo esta denominación genérica se engloban todas las acciones orientadas a corregir las situaciones que provocan que un SLA haya dejado de ser válida. Las acciones correctoras son dependientes del parámetro de la SLA que se ha incumplido, y de forma general incluye las dos acciones anteriores.

En esta enumeración se han excluido las operaciones de auto recuperación, puesto que estas se han delegado en el módulo de *Fault Tolerance*.

3.4.6. User Interface (UI)

El Interfaz de Usuario (UI) es el módulo encargado de comunicar a los usuarios de la plataforma con los componentes del sistema. El UI no forma parte en sí del despliegue de la nube (no es un componente perteneciente al clúster de componentes), sino que se trata de una herramienta externa. De esta forma, este componente no implementa ninguna funcionalidad por sí mismo, sino que actúa como puerta de enlace al sistema, interactuando con este mediante el envío de mensajes al resto de componentes que implementan los *core services*. Su objetivo es el de proporcionar una visión unificada de

la plataforma, y ocultar a los usuarios la complejidad de los componentes distribuidos.

El UI se compone de la parte cliente de los *core services* que exponen funciones al exterior, concretamente, las distintas variantes del *Catalog* y el módulo *Manager*, tal y como se ilustra en la Figura 3.7. Puesto que los *core services* van a ser implementados mediante servicios web, van a exponer funciones al exterior para que sean accedidas mediante SOAP, y por tanto los clientes que conectan con estas funciones públicas son los que conforman el UI de la plataforma.

En el caso de los servicios de *catalog*, las funciones públicas a las que se accederán mediante el UI son las de consulta (query), que permiten buscar servicios alojados en el sistema, recuperar su metadatos, obtener las referencias de las instancias desplegadas y conocer estadísticas sobre la plataforma y los servicios. Por su parte, en el caso del módulo *Manager*, las funciones a las que se proporcionará acceso mediante el UI son aquellas relacionadas con la administración de servicios, tales como el alta y baja de servicios, despliegue, repliegue y actualización de instancias.

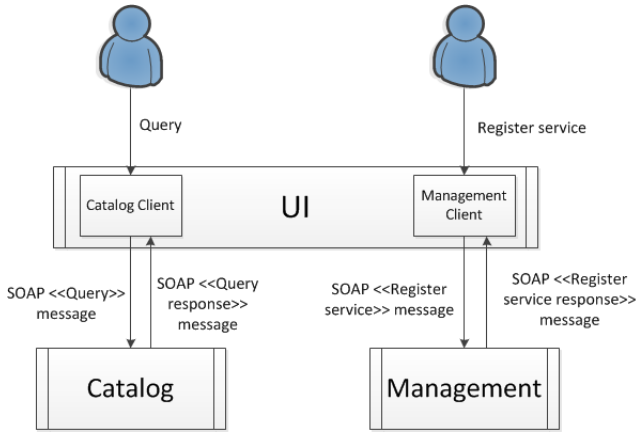

Figura 3.7: *Esquema de funcionamiento básico del componente UI. Los usuarios acceden a los clientes de los componentes para interactuar con la plataforma.*

Los clientes de los *core services* pueden implementarse de múltiples maneras, y pueden ser presentados como aplicación de línea de comandos, aplicación gráfica, aplicación web, etc.

3.4.7. SLA Manager

El *SLA Manager* es el módulo encargado de administrar los distintos SLA de la plataforma, es decir, comprobar la validez o cumplimiento de un SLA dentro de un sistema. La validez de un SLA consiste en comprobar que sus términos son correctos con respecto a otros SLA, o de forma general, a los términos de uso de la plataforma (que en sí mismos son un SLA). El cumplimiento de un SLA consiste en comprobar que los términos que define el SLA (por ejemplo tiempo de respuesta) se están cumpliendo en el instante actual. Estas dos operaciones sobre los SLA necesitan distintos requisitos y se implementan de distinta forma en la plataforma.

Este módulo es consultado por el *Manager* antes de realizar distintas operaciones, para validar tanto los SLA de los servicios como los de los usuarios. De forma adicional, el módulo de Quality of Service utiliza al *SLA Manager* para monitorizar el estado de la plataforma, comprobando de forma periódica que las distintas SLAs soportados por el sistema se cumplen.

Sus funciones básicas son las siguientes.

- Validar SLA de servicio: Comprueba que el SLA de un servicio permite realizar una acción determinada (por ejemplo registrar un servicio).
- Validar SLA de instancia: Comprueba que el SLA de una instancia permite realizar una acción determinada (por ejemplo desplegar una instancia).
- Validas SLA de usuario: Comprueba que el SLA de un usuario permite realizar una acción determinada (por ejemplo actualizar un servicio).
- Se cumple SLA de servicio: Comprueba que el SLA de un servicio se cumple en el instante actual.
- Se cumple SLA de instancia: Comprueba que el SLA de una instancia se cumple en el instante actual.
- Se cumple SLA de usuario: Comprueba que el SLA de un usuario se cumple en el instante actual.

3.4.8. Load Balancer

El módulo *Load Balancer* se encarga de administrar los recursos del sistema, realizando un equilibrado de carga entre ellos. Este módulo actúa como proxy ante peticiones de despliegue de servicio, despliegue de componentes y

acceso a instancias, ya que son las operaciones que tienen como resultado un recurso del sistema listo para ser utilizado. Sus funciones son las siguientes.

- Obtener contenedor virtual objetivo: Frente a una petición de despliegue de instancia, se debe determinar el *Virtual Container* donde se va a alojar la misma. Para elegir el destino, los contenedores disponibles son filtrados en base a el SLA del servicio, y se seleccionarán como candidatos únicamente aquellos que cumplen con la misma. En un segundo paso, se aplica un algoritmo de equilibrado de carga entre los candidatos para determinar el contenedor óptimo para albergar la instancia, que será devuelto como contenedor objetivo.

- Obtener nodo físico objetivo: Frente a una solicitud de despliegue de componente (típicamente un *Virtual Container*), es necesario averiguar el nodo físico donde va a ubicarse. Los nodos físicos representan el hardware sobre el que se despliega la plataforma, y por tanto su caracterización difiere de los componentes lógicos que conforman la solución, como son los *Virtual Container*. La elección del nodo físico debe someterse a unas exigencias equivalentes a las de la selección de un contenedor virtual, filtrando los nodos candidatos de entre los disponibles y escogiendo uno de ellos en base a un algoritmo de equilibrado de carga.

- Obtener endpoint reference objetivo: La solicitud de acceso a un servicio implica la necesidad de disponer de la endpoint reference de una instancia del servicio solicitado. En este caso, el *Load Balancer* actúa como proxy inverso frente a las peticiones de usuario externas, redirigiéndolas a los recursos virtuales utilizando un algoritmo de equilibrado de carga, considerando las posibles instancias redundantes que puedan existir.

- Algoritmo de equilibrado de carga: Esta es una función interna que debe encargarse de implementar el algoritmo de equilibrado de carga mencionado en todas las funciones anteriores. Esta función debe recibir como entrada un conjunto de candidatos (nodos físicos, *Virtual Containers*, endpoint references) junto con sus características, y seleccionar de entre todos ellos aquel recurso que produzca el mejor equilibrado en la carga computacional de la plataforma. Pueden existir múltiples implementaciones para este tipo de algoritmo, desde la más básica, como puede ser la selección aleatoria, hasta otras más elaboradas como la selección del recurso con menor carga actual o selección del recurso

con menor carga con respecto a su potencia. El desacoplamiento de esta función permite a los administradores modificar el algoritmo de equilibrado de carga de acuerdo a sus requerimientos, sin inferir con el comportamiento del resto de componentes.

El comportamiento general del módulo se muestra en la Figura 3.8, donde *Load Balancer* proporciona al módulo *Manager* la referencia a un *Virtual Container*.

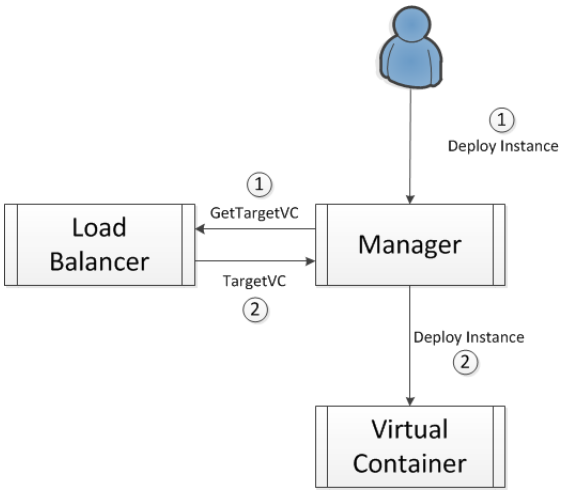

Figura 3.8: *Ejemplo de la provisión de un recurso por parte del módulo Load Balancer.*

3.4.9. Interoperator

El módulo de interoperabilidad es el encargado de implementar la capacidad de la plataforma de relacionarse con otras nubes. Una nube requiere interoperar con otra, entre otras situaciones, cuando sus recursos son insuficientes para llevar a cabo sus objetivos, bien sea mantener el nivel de calidad de servicio, bien sea cumplir con los SLA del sistema. Ante la falta de recursos, la acción inicial puede ser el auto escalado de la plataforma, pero una vez la infraestructura base de la nube se ha saturado, la capacidad de aumentar sus recursos queda limitada.

Una interacción entre nubes se concibe como un proceso de comunicación entre una plataforma consumidora y otra proveedora, cuyo objetivo es el poder disponer de recursos de otra solución para el uso propio. Este "alquiler" se define como la cesión temporal de recursos a cambio de un precio, que

debe contemplar no sólo un valor monetario, sino cualquier bien que sea definido como valioso por las plataformas (por ejemplo reputación, valoración positiva, créditos reembolsables, etc.).

Este módulo cuenta con las siguientes operaciones:

- Solicitar recursos: Esta operación recibe como entrada el SLA que indica las características de los recursos solicitados y unas restricciones sobre los mismos. Este módulo dispone de una serie de nubes candidatas registradas, es decir, nubes con las que se ha establecido un acuerdo de interoperabilidad, lo que implica el registro de la nube consumidora en la proveedora como usuaria (bajo el rol Cloud) y la definición de un SLA de interacción. De entre todas las nubes disponibles, el módulo selecciona las que cumplen con las restricciones especificadas por el SLA y finalmente elige como proveedora a aquella que ofrece el menor precio por los recursos solicitados. Una vez seleccionada la nube objetivo, se procede a ejecutar el algoritmo de negociado, que ha sido expuesto con detalle en el apartado (3.1.12). Utilizando este algoritmo, ambas nubes llegan a un acuerdo sobre la cantidad y precio de los recursos a alquilar. Este procedimiento se guía por el criterio de negociación de la nube consumidora, como pueden ser "mayor cantidad de recursos por precio fijo", "recursos fijos por el menor precio", "mejor relación recursos-precio", o distintos grados de flexibilidad en el negociado. En último lugar, cuando se ha llegado a un acuerdo, se procede a la cesión de los recursos de una nube a otra. Esta cesión debe otorgar a la nube consumidora la posibilidad de utilizar los recursos pero mantener la propiedad de la nube proveedora sobre estos. La aproximación utilizada para Cloud ComPaaS a esta operación es que el componente *Interoperator* actúe como proxy para las peticiones de despliegue y repliegue de componentes. Para ello, tras una negociación exitosa, en la nube proveedora se modifica el SLA de usuario de la nube consumidora para indicar que tiene derecho a consumir una cantidad de recursos determinada, y estos recursos se reservan en exclusiva para la nube consumidora. A partir de ese instante, la nube consumidora es libre de utilizar estos recursos a través de las operaciones que ofrece *Interoperator*.

- Desplegar *Virtual Container*: Recibe una petición de nube consumidora para desplegar un *Virtual Container*. El comportamiento de esta operación es análogo a la de "desplegar instancia" del módulo *Manager*. Tras realizar las comprobaciones de seguridad y SLA, si la nube consumidora está habilitada para utilizar recursos, se despliega el *Virtual*

Container y se anotan sus datos en el catálogo, asociada a la nube proveedora.

- Replegar *Virtual Container*: Recibe una petición de una nube consumidora para replegar un *Virtual Container*. Tras realizar las comprobaciones de seguridad, se verifica que el *Virtual Container* señalado pertenece a la nube consumidora y, en caso de que así sea, se procede a su repliegue.

- Revocar cesión: Recibe una petición de una nube consumidora para revocar una cesión en curso entre ella y el proveedor. Frente a esta revocación, se repliegan los *Virtual Container* alojados y se liberan los recursos reservados.

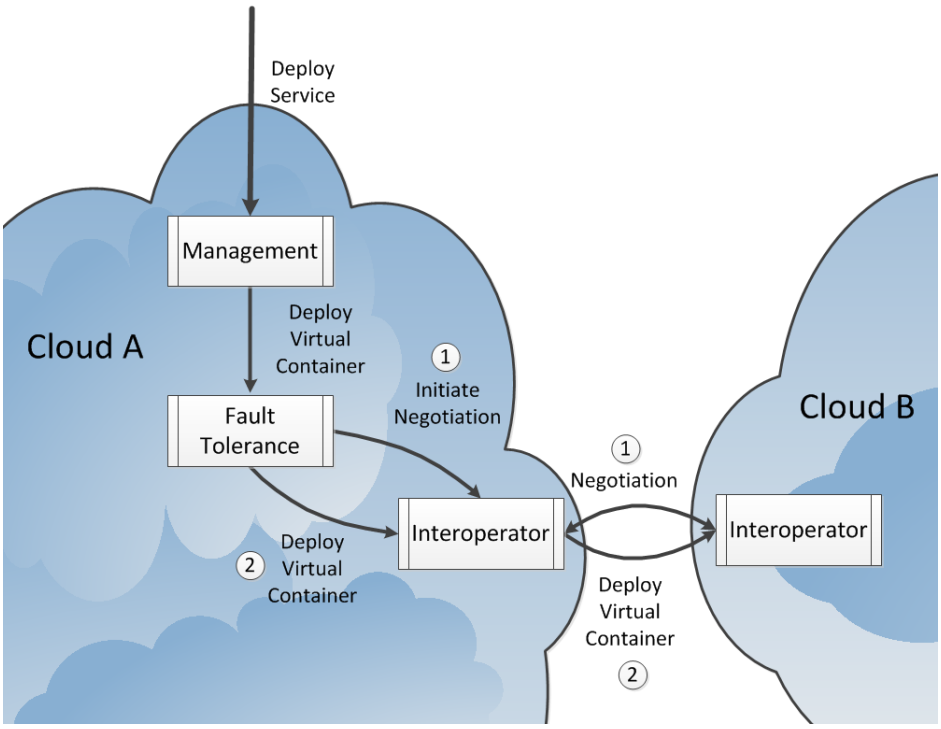

Figura 3.9: *Cuando la Cloud A no tiene suficiente recursos para atender una petición de despliegue de servicio, la nube acude a la interoperabilidad para obtener nuevos recursos. Tras una negociación satisfactoria, Cloud A puede desplegar Virtual Containers en la infraestructura de Cloud B.*

El esquema básico de funcionamiento del sistema se ilustra en la Figura 3.9. Cuando se lleva a cabo una cesión de recursos, esta se realiza de forma indefinida, y debe ser la nube consumidora la que anule la cesión cuando los recursos ya no son necesarios. Los términos de pago, que son especificados en el SLA de interacción, de forma habitual se definen en términos de pago por uso (cantidad de recursos y tiempo de utilización) de forma periódica. Este proceso de interacción se puede definir de forma relativamente inmediata para dos nubes que utilizan Cloud ComPaaS, y que hacen uso de los mismos componentes de interoperabilidad. Sin embargo, la inclusión de un sistema de plug-ins puede adaptar este procedimiento para funcionar con nubes públicas, formando una nube hibrida. Estos plug-ins representarían el papel de nube proveedora, y el proceso de negociación se limitaría a aceptar o rechazar la tarifa de los recursos del proveedor.

3.4.10. Fault Tolerance

El módulo de tolerancia a fallos es el encargado de proporcionar a la plataforma la capacidad de auto recuperación, así como manejar los supuestos de fallo y particionamiento de la plataforma. Como se ha comentado en la definición de la arquitectura, una instancia de una plataforma se compone de, al menos, una instancia de cada uno de los componentes de la misma. Esto implica que si, debido a fallos, la plataforma se queda sin instancias de alguno de sus componentes, no puede funcionar con normalidad. Adicionalmente cada plataforma dispone de una configuración de componentes, definida por los administradores, que indica el número de réplicas de cada componente, con lo cual si el número de réplicas es menor al requerido podemos considerar que la plataforma no está funcionando correctamente.

Cuando un componente abandona la plataforma, bien sea por decisión de la misma (auto escalado de disminución), por acción de los administradores (mantenimiento) o por fallo, la labor del módulo de tolerancia a fallos consiste en comprobar que se siguen cumpliendo las condiciones de funcionamiento de la plataforma especificadas anteriormente, y realizar las acciones correctivas para restablecer el correcto funcionamiento. Por otro lado, debido a la naturaleza de las operaciones del módulo de despliegue y eliminación de nodos de la plataforma, estas pueden ser utilizadas para implementar la funcionalidad de elasticidad y auto escalado del sistema, proporcionando un medio para aumentar o disminuir el número de elementos en ejecución.

Las funciones del módulo de *Fault Tolerance* son las siguientes.

- Comprobación de estado: Esta es una función que se ejecuta de forma

autónoma periódicamente, con el objetivo de comprobar el estado de la plataforma. Esta verificación del estado hace referencia a la comprobación del número de réplicas de cada componente desplegadas, con respecto a la configuración de la plataforma, así como la verificación del estado de cada una de ellas.

- Replegar componente: Esta función se invoca en caso de que se detecte un componente fallido o con funcionalidad de degradada. El módulo se encarga de detener la ejecución del componente especificado así como de eliminar su información de la plataforma. Adicionalmente se puede utilizar esta función para replegar *Virtual Containers* que ya no son necesarios, proporcionando la característica de downscaling.

- Desplegar componente: Esta función se invoca cuando el número de réplicas de un componente es menor del estipulado en la configuración de la plataforma. Para realizar el despliegue, el módulo selecciona un nodo físico donde desplegar el nuevo componente, recupera el binario del componente de un repositorio (el catálogo) y realiza la puesta en funcionamiento y el registro de los datos del nuevo componente en la plataforma. En caso de que la plataforma no disponga de suficientes nodos físicos como para realizar el despliegue de un nuevo componente, se acude al módulo de interoperabilidad para solicitar a otras nubes los recursos necesarios. Si finalmente no se pueden obtener nuevos recursos para alojar el componente o instancia requerido, la función falla. Adicionalmente, esta función puede ser utilizada para desplegar nuevos *Virtual Container* cuando es necesario añadir recursos al sistema, proporcionando la característica de upscaling.

Capítulo 4

Prototipo

Para comprobar la validez de los principios recogidos en este documento y la corrección de la arquitectura y diseño elaborados, se ha desarrollado un prototipo que sirve de prueba de concepto de los mismos.

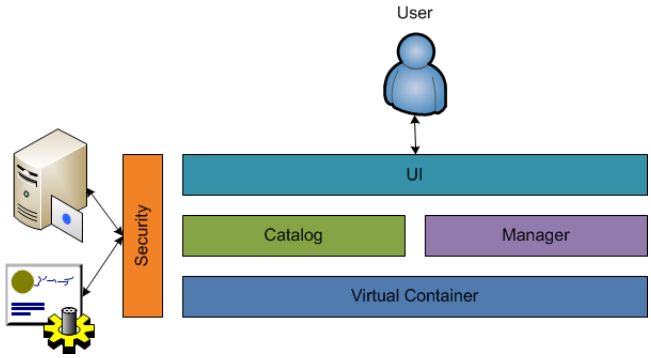

Figura 4.1: *Arquitectura reducida del prototipo.*

A la hora de definir un prototipo de la plataforma, se ha seleccionado un subconjunto de la arquitectura que resulta significativo para establecer la validez de los conceptos definidos para la plataforma, y que representa con suficiente amplitud las características de la misma. Este prototipo sirve de punto de partida para futuros desarrollos que incluyan la totalidad de la plataforma.

Los componentes seleccionados se presentan en la Figura 4.1.

La arquitectura del prototipo se compone de los *Virtual Container* junto con los módulos *Manager* y *Catalog*, un marco de seguridad y un interfaz de usuario. El objetivo del prototipo es ofrecer una prueba de concepto de la plataforma en términos de despliegue, administración y descubrimiento de

servicios. El resto de módulos se han excluido del prototipo debido a que su implementación conllevaría una cantidad de tiempo y esfuerzo excesivo para el ámbito de la Tesis de Máster. Por otro lado los componentes seleccionados conforman la funcionalidad básica de la plataforma, y el funcionamiento de otros módulos carece de sentido sin dichas funcionalidades básicas.

La capa de abstracción de almacenamiento no se ha añadido al prototipo debido a la complejidad de implementación de una solución de almacenamiento Cloud. La línea del proyecto en almacenamiento Cloud es la de utilizar una solución de almacenamiento basada en estándares que están siendo definidos en la actualidad [46].

Los módulos *Interoperator* y *SLA Manager* se han excluido del prototipo debido a la complejidad de los fundamentos teóricos de ambos módulos. Su desarrollo implica realizar extensos estudios y especificaciones de los elementos, protocolos y algoritmos implicados en cada uno, tras lo cual es necesario realizar la implementación del prototipo.

El módulo *Fault Tolerance* se ha excluido del desarrollo debido a la complejidad de la definición e implementación de protocolos de auto recuperación, así como la implementación de operaciones que permiten desplegar nuevos *Virtual Container* de forma automática. El módulo de Quality of Service por su parte, no ha sido implementado puesto que hace uso de todos los módulos anteriormente mencionados, y por tanto no puede funcionar de forma adecuada sin ellos.

Finalmente el módulo *Load Balancer* ha sido reducido e integrado en el módulo *Manager*, con el objetivo de reducir la complejidad y el tiempo de desarrollo del prototipo.

El estado actual del prototipo contiene las siguientes características:

- Un contenedor virtual capaz de alojar servicios web escritos en Java, desplegar y replegar servicios.
- Un UI desde el cual consultar información del catálogo y realizar acciones en el módulo de administración. Este UI se ofrece en formato web y por línea de comandos.
- Un catálogo persistente. Este catálogo está organizado como bases de datos replicadas entre todos los nodos.
- Un catálogo transitorio. Este catálogo tiene una organización distribuida y k-replicada. Su información se actualiza mediante la acción de los otros componentes, y es accesible para ser consultada desde el exterior mediante el UI.

- Un módulo de administración, que permitiese interactuar con la plataforma desde el UI.
- Se diferencia el concepto de servicio e instancia. Los servicios se registran en la plataforma, mientras que sus instancias son desplegadas de forma independiente.
- Un marco de seguridad básico. Se utiliza la autentificación de los usuarios mediante username-token.
- Concepto de versión. Los servicios poseen un número de versión que debe ser administrado por los distintos componentes. El contenedor virtual incluye la operación de actualizar servicio.
- Asegurado de la herramienta de clustering, requiriendo a los nodos disponer de un certificado X509 para ingresar en el clúster. Las comunicaciones intra-clúster están cifradas con dicho certificado.

4.1. Herramientas utilizadas

4.1.1. Contenedor de servicios: Apache Axis2

Puesto que la plataforma está orientada a la ejecución de Servicios Web Java, es necesario disponer de un *runtime* que permita alojar este tipo de servicios. Estas herramientas se denominan comúnmente "contenedores de servicios", y permiten el alojamiento y ejecución de servicios web, exponiendo su funcionalidad al exterior mediante una URL como un servidor web.

A la hora de elegir el contenedor de servicios que actuará como *runtime* virtual de la plataforma, se han seguido los criterios que se exponen a continuación:

- Soporte a servicios web Java: Puesto que, por cuestiones de facilidad de mantenimiento y de programación, los componentes del prototipo de la plataforma van a estar implementados utilizando el lenguaje Java, es necesario que el contenedor de soporte a este tipo de servicios.
- De libre distribución y código abierto: Para poder implementar algunas de las funcionalidades requeridas en la plataforma, será necesario tener acceso y enriquecer el código fuente de los contenedores. Por ello se requiere un contenedor de código abierto que permita su modificación y redistribución sin coste, dado que pretendemos que esta sea una plataforma que pueda ser utilizar por los investigadores.

- Portable: La portabilidad de la herramienta permitirá virtualizar la mayor cantidad de hardware, favoreciendo, por tanto, el alcance y distribución de la plataforma.

- Standalone: El contenedor debe ejecutarse de forma autónoma, incluyendo todas las librerías y elementos necesarios para su correcto funcionamiento, sin necesidad de software externo. Esta característica permitirá simplificar el proceso de distribución del mismo.

- Ligero: Una de las ventajas de la virtualización de servicios mediante plataformas PaaS cómo alternativa a IaaS es la reducción de la sobrecarga introducida por las máquinas virtuales. Así será importante reducir el consumo de recursos de este componente, para que no suponga una sobrecarga adicional que reduzca las ventajas de la plataforma frente a la utilización de máquinas virtuales.

Algunas alternativas barajadas han sido Apache CXF, Metro, Apache Axis y Apache Axis2 [47], puesto que cumplen con muchas de las características deseadas para este componente. Apache CFX proporciona un framework para servicios web que puede ser incluido en Tomcat o Spring, mientras que Metro es una implementación del 'stack' de servicios web para GlassFish. A pesar de ser interesantes, ambas han sido descartadas como opciones puesto que no pueden ser utilizadas sin la necesidad de software complementario. Por su parte, Apache Axis es un contenedor de Servicios Web Java con soporte a SOAP, que puede ser ejecutado tanto standalone como plug-in de Tomcat. Sin embargo, el desarrollo de Apache Axis se interrumpió en 2006, al ser sustituido por su sucesor, Apache Axis2.

Finalmente el contenedor de servicios elegido ha sido Apache Axis2, que es un contenedor de servicios web Java con dos implementaciones, una en Java (Axis2/Java) y otra en C (Axis2/C). Este contenedor redefine la arquitectura de Apache Axis, ofreciendo mayor flexibilidad, eficiencia y posibilidades de configuración que su antecesor. Además, Axis2 soporta SOAP 1.1, SOAP 1.2 y REST, y mediante un diseño modular permite la inclusión de plug-ins que expanden su funcionalidad admitiendo, por ejemplo, los estándares WS-* como WS-Security o WS-Addressing.

Axis2 se distribuye tanto en formato binario como en código fuente, y dicha distribución incluye todos los componentes básicos necesarios para ejecutar el contenedor y alojar servicios de forma autónoma. Adicionalmente existe una distribución .WAR (Web Archive) que se puede ejecutar desplegándose en un contenedor de Servlets (por ejemplo Apache Tomcat) [48].

Este contenedor de servicios cumple, por tanto, con todos los requisitos necesarios para nuestro proyecto y, adicionalmente, ofrece herramientas útiles para el mismo. Algunas de estas son un API para administrar la configuración del contenedor de forma programática, una herramienta para generar WSDL a partir del código de un servicio (code2wsdl), y otra para generar código a partir de un fichero WSDL (wsdl2code). Este último comando permite generar código tanto del cliente (stub) como servidor (skeleton) de un servicio determinado partiendo del documento WSDL del mismo, el cual a su vez puede ser generado automáticamente a partir del código del servidor. Utilizando ambas herramientas se puede automatizar la generación del código cliente de un servicio, partiendo del servidor del mismo.

Axis2, como todos los productos de la Apache Software Foundation, está publicado bajo la licencia Apache License 2.0 [49]. Este modelo de licencia permite el utilización, modificación y distribución para cualquier tipo de uso del producto software, y no obliga a que las obras derivadas utilicen la misma licencia, aunque sí que requiere que todos los derechos, patentes, marca registrada, etc. utilizados en el producto original se mantengan en el derivado y que se identifique de forma clara aquellos ficheros que hayan sido modificados, en caso de que esto ocurra.

Framework seguridad: Apache Rampart

Apache Rampart [50] es el framework de seguridad de Apache Axis2, y soporta varios estándares como WS-Security, WS-Trust y tokens SAML, permitiendo características como la autenticación, la integridad, la confidencialidad y el no repudio. Para el prototipo implementado se ha hecho uso únicamente de parte del estándar WS-Security, y más concretamente, la autentificación mediante username-token. La utilización de Apache Rampart permite refinar los mecanismos de seguridad en futuros desarrollos.

Mediante Rapart y haciendo uso de WS-SecurityPolicy [51], se puede establecer la política de seguridad que deben seguir los servicios web. Para ello se ha de preparar un fichero que permita indicar a los clientes y servidores el protocolo de seguridad que van a seguir las comunicaciones entre ambos y, por tanto, proporciona al framework las directivas necesarias para asegurar y validad los mensajes SOAP intercambiados. A continuación se muestra un ejemplo del contenido de este tipo de fichero. En el cliente se indica que las acciones de salida deben incluir un sistema de autentificación por username-token, mientras que en el servidor se indica que las acciones de entrada van a estar autentificadas por username-token, y que la clase encargada de comprobar las contraseñas es cloud.paas.security.server.PWCBHandler. Pa-

ra cambiar el esquema de seguridad de username-token a otros habría que cambiar el contenido de las etiquetas "action" por el método deseado (por ejemplo certificados X509) e indicar el flujo de acciones a llevar a cabo (por ejemplo firmar contenido, cifrar contenido, añadir timestamp, etc.).

```
Servidor
<parameter name=''InflowSecurity''>
     <action>
         <items>UsernameToken</items>
         <passwordCallbackClass>
         cloud.paas.security.server.PWCBHandler
         </passwordCallbackClass>
     </action>
</parameter>

Cliente
<parameter name=''OutflowSecurity''>
     <action>
         <items>UsernameToken</items>
     </action>
</parameter>
```

Script 1: *Política de seguridad mediante Username-token.*

Dependiendo de la política de seguridad utilizada, en ocasiones es necesaria la inclusión de elementos extra como son almacenes de claves, certificados, los ficheros de configuración de los mismos o modificaciones en el código fuente (por ejemplo clase PasswordCallback) en los clientes y servidores. Al margen de estos elementos, el proceso de asegurado y validación de los mensajes se realiza de forma transparente para el usuario de este entorno.

Apache Rampart, como producto de la Apache Software Foundation, utiliza la licencia Apache License 2.0, cuyas características han sido mencionadas en el apartado 4.1.1.

4.1.2. Sistema Gestor de Bases de datos: HSQLDB

Para la implementación de los catálogos de la plataforma es necesario disponer de un mecanismo que le permita almacenar, manipular y recuperar información de forma simple, eficiente y estandarizada. Debido a estas

necesidades, la opción más adecuada resulta la utilización de bases de datos como mecanismo eficiente para almacenar y organizar la información.

Las características que buscamos en un Sistema Gestor de Bases de Datos (SGDB), para que sea aplicable a la plataforma, son las siguientes.

- Ligero: Que consuma pocos recursos del sistema, con el fin de que no introduzca sobrecargas excesivas.
- Modelo de licencia: Que tenga una licencia compatible con el resto de licencias utilizadas y, en especial, que permita la distribución siguiendo el modelo de código abierto para no condicionar la plataforma.
- Redistribuible: Que sea fácilmente redistribuible y no requiera instalaciones ni configuraciones complejas.
- Portable: Que se pueda utilizar con independencia del hardware subyacente.
- Integración: Que la base de datos pueda ser embebida en cada aplicación, volviéndose independiente de componentes externos.
- Base de datos en memoria: Que permita la posibilidad de utilizar in-memory database. Esta modalidad aporta un mayor rendimiento a costa de no ofrecer persistencia.
- Backups: Que permita generar copias de seguridad de bases de datos existentes, o recuperar una base de datos a partir de un fichero de copia de seguridad.

Las alternativas barajadas han sido MySQL, Apache Derby, SQLite, H2 y HSQLDB [52]. De entre estas, MySQL muestra, en general, los peores resultados de rendimiento en distintos benchmarks realizados [53], además de que requiere instalación y configuración potencialmente compleja. Por su parte, H2 no permite la creación y recuperación de copias de seguridad, con lo cual ha sido descartada como opción. SQLite y MySQL están escritos en C, y requieren un ejecutable distinto para cada tipo de Sistema Operativo donde se van a ejecutar, con lo cual también han sido descartados.

De las dos herramientas restantes (Apache Derby y HSQLDB), la opción seleccionada ha sido HSQLDB, por presentar los mejores resultados en varios de los benchmark consultados y por ofrecer todas las características requeridas por la plataforma.

HSQLDB es un sistema gestor de bases de datos (SGDB) de código abierto escrito en Java, que se distribuye como una única librería .jar y

que puede ser embebido en una aplicación Java, proporcionándole acceso a bases de datos relacionales SQL.

Debido a la naturaleza transitoria de parte del catálogo de la plataforma, la opción de in-memory database de esta herramienta encaja perfectamente con los requisitos de la misma ya que permite gestionar una base de datos en memoria, sin necesidad de que tenga un reflejo en el disco. Del mismo modo, manteniendo una base de datos solamente en memoria, se mejoran las prestaciones del catálogo y se evita la generación de ficheros innecesarios en la máquina.

Por otro lado, la opción de copia de seguridad facilita la administración del catálogo persistente del sistema. Los nuevos nodos que se añaden al catálogo tienen la necesidad de recuperar el estado de la base de datos para poder sincronizar su comportamiento con el del resto. Utilizando copias de seguridad, los nodos pueden restaurar el estado de la base de datos y sincronizarse con el resto de catálogos, simplificando y acelerando la puesta en marcha de nuevas réplicas.

HSQLDB se distribuye bajo una licencia propia derivada de la licencia BSD, la cual permite la utilización, modificación y distribución del producto para cualquier tipo de uso siempre y cuando se mantengan los derechos sobre el producto original, se distribuya el documento de la licencia junto con el mismo y no se utilice el nombre de HSQL Development Group o el de cualquiera de sus colaboradores para promocionar el producto derivado, a excepción de que se disponga de una autorización por escrito para ello.

4.1.3. Clustering: JGroups

Puesto que la plataforma se compone de nodos físicamente distribuidos, es necesario agrupar estos elementos en un clúster lógico en que todos se conozcan entre sí y sean capaces de comunicarse. Mediante la habilitación de un sistema de comunicaciones interno, podemos garantizar una vía segura, fiable y eficiente de intercambiar mensajes internos entre componentes del sistema.

En la búsqueda de herramientas de este tipo únicamente se ha encontrado una que presenta estas características para el lenguaje de programación Java, el toolkit JGroups [54]. Esta herramienta permite realizar comunicaciones fiables multipunto, e incluye una serie de características adicionales que se enumeran a continuación:

- Creación de grupos de procesos: Utilizando JGroups se pueden generar grupos de procesos, donde todos los miembros se federan en un clúster.

- Entrada y salida de miembros: La configuración de los grupos es dinámica, pudiendo entrar o salir miembros a lo largo del tiempo de vida del mismo.

- Membresía: Todos los miembros del grupo tienen una identidad única y se conocen entre sí. La configuración del grupo en un instante determinado se denomina vista. JGroups asegura que la vista es consistente entre todos los miembros del grupo, realizando cambios de vista cuando nodos se unen o abandonan el grupo.

- Detección de fallos: JGroups detecta y elimina procesos fallidos del sistema. Esta detección se realiza utilizando un protocolo de 'sospecha' sobre el fallo de procesos. En el caso de que algún miembro sospeche del fallo de otro, todos acuerdan realizar un cambio de vista y eliminar al proceso del grupo.

- Envío y recepción de mensajes punto-a-grupo (multipunto).

- Envío y recepción de mensajes punto-a-punto.

- Envío de mensajes fiable y entrega ordenada. Se asegura que todo mensaje es entregado a todos sus destinatarios o bien a ninguno de ellos (atomicidad), y que dicha entrega respeta un determinado orden (FIFO, causal, total).

JGroups se compone de una pila de protocolos, cada uno de los cuales rige un aspecto distinto del comportamiento del mismo. Estos protocolos pueden ser modificados o sustituidos sin afectar al comportamiento del resto de componentes de la pila, otorgando a la herramienta la suficiente flexibilidad como para poder adaptarse a gran cantidad de situaciones.

Este producto proporciona múltiples ventajas para un sistema compuesto por nodos físicamente distribuidos y que deben comunicarse entre sí, como es el caso de nuestra plataforma. JGroups permite agrupar los nodos heterogéneos, ofrecer una visión consistente del grupo para cada elemento, notificar del fallo de elementos, cifrar las comunicaciones, proporcionar entrega de mensajes fiable, etc.

JGroups hace uso de la licencia GNU Lesser General Public License, o LGPL. Esta licencia permite la utilización y distribución del producto software como una librería, utilizada por un producto externo, pudiendo ser este utilizado para cualquier fin, siempre y cuando se mantengan los derechos del mismo. Sin embargo, en caso de utilizar versiones modificadas del producto

o generar un producto derivado, esta versión modificada o producto derivado debe ser publicado necesariamente bajo la misma licencia LGPL.

4.2. Implementación

En esta sección se va a proceder a exponer los detalles de implementación que debido a su complejidad, relevancia o implicaciones se considera que deben ser resaltados en este documento.

4.2.1. Diseño general

Los componentes que gestionan el catálogo, los dedicados a la administración de la plataforma y el contenedor virtual han sido implementados como servicios web escritos en Java, para el contenedor de servicios Apache Axis2. Por otro lado, el componente encargado de proporcionar el interfaz de usuario (UI) se ha implementado como un conjunto de páginas web en JSP alojadas en Apache Tomcat. Adicionalmente se han elaborado unos clientes Java, donde los parámetros están embebidos en el código fuente, de cara al testeo de la plataforma en un entorno programático.

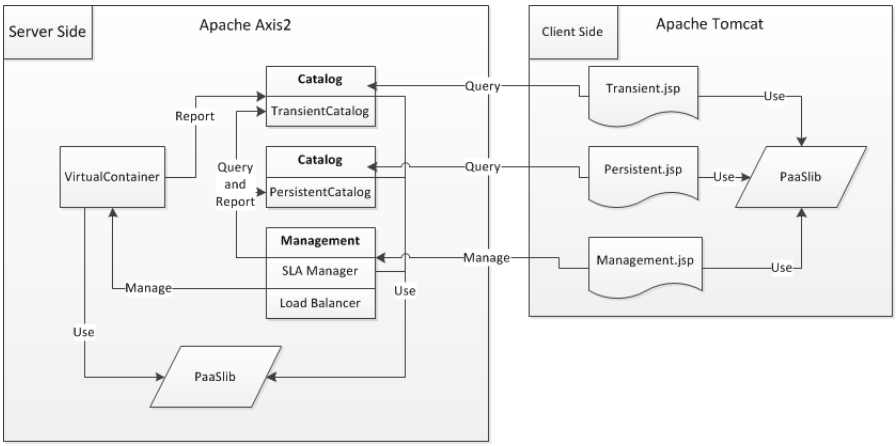

Figura 4.2: *Esquema básico de la implementación de la plataforma.*

La capa de seguridad se ha encapsulado, junto con otros elementos comunes a todos los componentes, en una librería denominada PaaSlib.jar. De esta forma, en el mismo paquete, se incluyen las funciones comunes, que son utilizados por los cuatro componentes principales de la plataforma: catálogo (con sus dos variantes), contenedor virtual, administración e interfaz de

usuario.

Por su parte, la implementación del catálogo se ha dividido en dos, de acuerdo a las características de persistencia. Ello ha dado lugar a los servicios web "TransientCatalog" y "PersistentCatalog", que implementan la parte transitoria y persistente del catálogo, respectivamente. Estos componentes pueden considerarse como subcomponentes del catálogo, y se encargan de implementar el catálogo de instancias (transitorio) y al catálogo de servicios (persistente).

La Figura 4.2 muestra un esquema de los componentes descritos, así como las principales relaciones entre ellos. En los siguientes apartados se describen con más detalle los aspectos más relevantes de estos componentes.

4.2.2. Librería PaaSlib

En el diseño de la plataforma se ha establecido que todas las características comunes a los componentes (seguridad, almacenamiento, clustering) se van a implementar como librerías. De este modo se saca mayor partido de la modularidad, desacoplamiento y reusabilidad de los mismos. En la implementación del prototipo, estos elementos comunes han sido recogidos en una única librería por una cuestión de simplicidad ya que, al tiempo, favorece la redistribución de la misma. Sin embargo, es posible que este criterio varíe en caso de que se continúe con el desarrollo de las características de la plataforma, dado que potencialmente aumentará la complejidad de estos elementos y por tanto sea conveniente factorizarlos en distintas librerías específicas.

Las funcionalidades comunes que implementa la librería se describen a continuación.

QueryResponse

La librería incluye la clase respuesta a las consultas del catálogo. Esta clase representa la respuesta de una consulta, y se compone bien de un conjunto de objetos (de clase variable) o bien de un conjunto de excepciones, o incluso de una mezcla de ambos. Se ha implementado como un elemento común puesto que múltiples componentes de la plataforma realizan consultas al catálogo, y por tanto reciben como respuesta objetos de ésta clase.

Clases comunes

En esta librería se ha creado un paquete, que hemos denominado "components", y que incluyen una serie de clases comunes que son utilizadas por todos los módulos. Además este paquete también incluye enumeraciones que

igualmente son utilizadas en la implementación de varios componentes. Las clases incluidas en este paquete son las siguientes:

- Component: Clase de la que heredan todos los componentes del sistema. Esta clase incluye una instancia de la clase Group que, como se verá más adelante, es utilizada para establecer las agrupaciones en clústeres.
- User: Representa un usuario en el sistema. En esta versión, esta clase resulta relativamente sencilla, pues sólo incluye el nombre de usuario y ciertos metadatos como la contraseña o el rol de usuario. En la versión completa de la plataforma esta clase debe incluir aspectos como el SLA de usuario y otros metadatos adicionales.
- SLA: Representa un SLA en el sistema. En la actualidad, este SLA incluyen solamente valores concernientes a servicios e instancias, como CPU, memoria o ancho de banda. Estos valores representan requisitos de los servicios para su despliegue.
- Service: Es una clase que representa un servicio en el sistema, e incluye todos los metadatos asociados a la representación de un servicio en la plataforma.

Las enumeraciones incluidas en esta librería son las siguientes:

- Auth: Enumera los métodos de autentificación disponibles. En la actualidad la plataforma solamente da soporta a autenticación nula o mediante username-token, pero la utilización de esta enumeración favorecer su extensibilidad.
- ComponentCode: Enumera los tipos de componentes disponibles en la implementación de la plataforma.
- Role: Enumera los roles de usuario disponibles en la plataforma. Estos roles hacen referencia a los grupos principales establecidos en el apartado3.4.1.

Marco de seguridad

Las operaciones y clases asociadas al marco de seguridad se encuentran incluidas en esta librería. En concreto se ha implementado un esquema de seguridad basado en username-token, donde los usuarios se identifican en el

sistema mediante un par usuario-contraseña. Se han dividido las operaciones en aquellas que están orientadas a los clientes y las que utilizará el servidor.

En concreto, para los usuarios se proporciona la clase ClientCredentials. Esta clase actúa como envoltorio a las credenciales de usuario, y sirve para la autentificación ante los *core services*. La clase ofrece la función setUsernameToken, mediante la cual los usuarios proporcionan su nombre de usuario y contraseña.

En el lado servidor, la autentificación se lleva a cabo en una clase PWCBHandler, la cual es invocada para comprobar las credenciales de un usuario. Esta clase comprueba que en las credenciales almacenadas en el servidor el nombre de usuario recibido se corresponde con la contraseña proporcionada. Esta es una clase de tipo "callback", y es invocada de forma automática por Apache Rampart cuando se realiza una petición sobre un servicio asegurado mediante username-token. En caso de que la verificación sea positiva, se procede a invocar al servicio solicitado, y en caso de que falle, se devuelve al usuario una excepción indicando que la autentificación ha fallado.

En el servidor, la clase auxiliar UserRegister obtiene los metadatos de un usuario del sistema. Esta clase permite, a partir del nombre de usuario autenticado, obtener los metadatos de un cliente identificado por su nombre, obtener los metadatos del cliente 'entrante' (es decir, el cliente que está llevando a cabo la interacción desde la cual se invoca el método) y delegar las credenciales del cliente entrante para una nueva interacción de salida. Esta clase hace uso del registro de los usuarios almacenados en la plataforma para obtener los metadatos de los mismos.

En el prototipo, los registros de usuario han sido codificados en el código fuente por cuestión de simplicidad, dado su carácter de prototipo. Sin embargo, se ha incluido la implementación con el objetivo de contemplar la casuística introducida por este componente.

Clustering

La agrupación de los componentes de la plataforma se realiza mediante la librería JGroups. Para facilitar la interacción con dicha librería, se ha creado un paquete (al que hemos denominado "util.groups"), en el que se ha implementado la clase Group, que actúa como envoltorio de las operaciones de comunicaciones a grupos de dicha librería. Esta clase, al instanciarse, recibe el nombre del grupo al que debe unirse, el tipo de componente que la invoca y una referencia al mismo.

Esta clase permite enviar mensajes de tipo *unicast* o de tipo *multicast* al grupo, y procesar los mensajes recibidos. Además cuenta con una instancia de

la clase "ComponentList", una clase que hemos implementado para permitir mantener una vista de los componentes que se encuentran dentro del grupo. En realidad esta clase es una lista que asocia el identificador de cada nodo del grupo a un ComponentCode para tener una información rápida de la composición básica del grupo.

Se ha incluido en la librería el interfaz ViewUpdater, para permitir a las clases que lo implementan recibir notificaciones cuando se produce un cambio de vista en el grupo. El interfaz incluye dos operaciones (addComponents y removeComponents), que indican que un conjunto de componentes se han añadido o han abandonado el grupo, respectivamente. De esta forma, cuando se producen cambios en un grupo, la clase Groups correspondiente lo notifica a todas las instancias de ViewUpdater que tiene suscritas, mediante la invocación de dichos métodos, avisando así acerca de los cambios que se han producido en el grupo.

Comunicaciones intra-clúster

Las comunicaciones entre componentes de la plataforma se realizan mediante intercambio de mensajes a través de la librería JGroups. Para implementar estas comunicaciones, se han utilizado un conjunto de clases que conforman los mensajes y dos proxies, uno de recepción (Receiver) y otro de emisión (Sender). A su vez, este segundo proxy se han dividido en dos variantes, según el mensaje sea una consulta (Request) o una respuesta (Response). De esta forma, los componentes envían mensajes de consulta (RequestSender) o respuesta (ResponseSender) y reciben los mensajes en el Receiver.

La diferencia entre mensajes de consulta y mensajes de respuesta es, básicamente, que los mensajes de consulta bloquean al emisor esperando una respuesta, mientras que las respuestas no son bloqueantes. Ante una consulta, el emisor se bloquea en una espera hasta que se recibe una respuesta o bien salta un timeout que determina que no se ha recibido una respuesta dentro de un plazo de tiempo razonable (lanzando una excepción en este caso). Cuando un mensaje es recibido, el "Receiver" debe redirigirlo a su destino. Para ello, en primer lugar debe determinar si se trata de un mensaje de respuesta o de consulta. Si se trata de una respuesta, busca entre las consultas pendientes aquella con la que se corresponde, enlazando la respuesta con la consulta y notificando al hilo de ejecución bloqueado que la consulta ha finalizado.

Si el mensaje recibido es una consulta, se verifica que sea una consulta válida (es decir, que esté dirigida al componente donde se ha recibido) y si la consulta realizada en sí es válida (llamada a función, parámetros). En caso de ser válida, esta se redirige a la clase encargada de procesar consultas

para ese componente, donde mediante reflexión Java (java.lang.reflection) se realizan llamadas a funciones del componente, generando al finalizar el correspondiente mensaje de respuesta.

Todo el procedimiento de recepción de mensajes y envío de respuestas se realiza de forma automática por los receivers, así que las únicas operaciones que deben llevar a cabo los componentes es enviar consultas, con todas sus operaciones asociadas como son la generación del mensaje, el envío a través del grupo, el bloqueo de respuesta, etc. Para ocultar está complejidad, se han incluido en la librería una serie de subclases de "RequestSender"que realizan dichas operaciones de forma transparente, ofreciendo al componente una abstracción del envío simulando el estilo de las RPC.

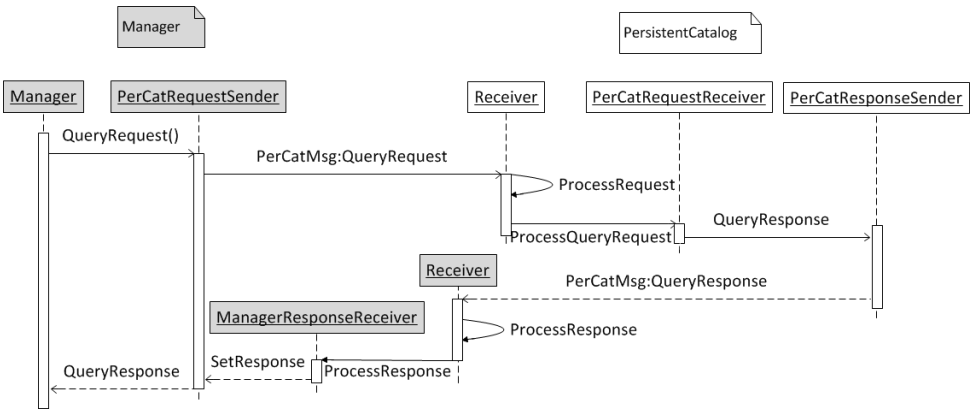

Figura 4.3: *Manager enviando una consulta al PersistentCatalog, a través del sistema de comunicaciones interno.*

El proceso de comunicación intra-clúster se ilustra con un ejemplo, en la Figura 4.3. En la figura, el componente *Manager* realiza una query al módulo PersistentCatalog. Para ello genera una instancia de la clase PersistentCatalogRequestSender, e invoca al método QueryRequest sobre dicha instancia. La clase PersistentCatalogRequestSender envía un mensaje de QueryRequest, y queda suspendido a espera de la respuesta. El mensaje de QueryRequest es procesado por la clase Receiver de PersistentCatalog, que tras identificar que se trata de una consulta (Request), genera una instancia de PersistentCatalogRequestReceiver, y le delega el mensaje para que lo procese. Esta clase procesa el mensaje, generando las llamadas necesarias al PersistentCatalog (no ilustrado en la figura, para facilitar la legibilidad de la misma) y finalmente genera una queryResponse. Para devolver el resultado, genera una instancia de PersistentCatalogResponseSender e invoca el méto-

do QueryResponse sobre él. Esta clase genera un mensaje QueryResponse que es enviado de vuelta al módulo *Manager* que originó la consulta, el cual lo procesa utilizando la clase Reciever. Una vez determina que el mensaje entrante se trata de una respuesta (Response), Receiver genera una instancia de la clase ManagerResponseReceiver y delega en ella el procesamiento de la respuesta. Esta clase extrae el contenido del mensaje, la asocia a la consulta enviada anteriormente y notifica al hilo de ejecución de la consulta de que ésta ha finalizado.

Catalog

Los catálogos del sistema se conciben como componentes que, además de forman un clúster con el resto de la plataforma, forma un clúster entre sí en una organización P2P. El catálogo se divide en dos partes, como se ha definido en la arquitectura, en un caso para recoger la información de carácter persistente y en el otro la transitoria, con las particularidades de cada uno de estos modelos.

Debido a esta especificación, y siguiendo el principio de modularidad propuesto para el desarrollo, nos hemos planteado que el diseño del catálogo ofrezca la flexibilidad suficiente como para permitir generar nuevos catálogos de forma sencilla. Para ello se han derivado las características comunes de ellos, y se ha separado la implementación en bloques funcionales.

Así, esta implementación se ha separado en los tres bloques lógicos definidos en la especificación de los mismos: el clustering, el *front end* y las bases de datos. De forma resumida, el bloque de clustering se encarga de administrar la pertenencia a grupos de los mismos. Por su lado, el bloque de *front end* se encarga de implementar la lógica de los catálogos, como son sus operaciones o la puesta en funcionamiento de los mismos, haciendo distinción entre el caso persistente y el transitorio. Finalmente, el bloque de bases de datos administra la base de datos embebida donde en última instancia van a ser almacenados los datos.

La implementación de los servicios de catálogo se ha organizado siguiendo una estructura jerárquica de clases desacopladas, lo que permite una gran flexibilidad a la hora de definir y crear nuevos catálogos. A continuación se explican los detalles de esta organización y las partes que pueden ser modificadas para proporcionar el catálogo concreto. En nuestro caso se ha realizado una implementación orientada a las características concretas de los servicios virtuales en la plataforma.

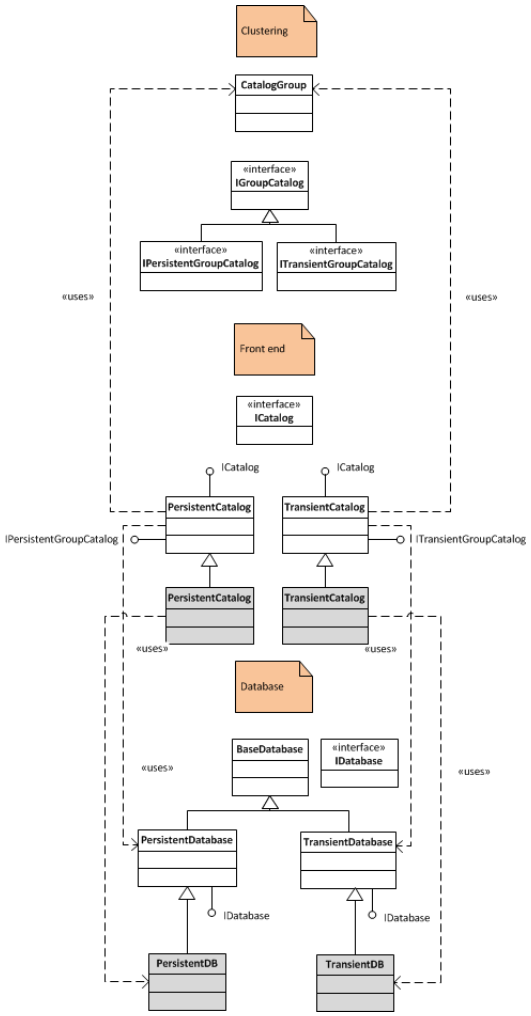

Figura 4.4: *Jerarquía de clases del componente catálogo. En gris se destacan las clases que conforman la realización de los catálogos persistente y transitorio.*

Como se puede observar en la Figura 4.4, se ha creado un conjunto de clases implementan la funcionalidad básica de cada bloque:

- En el bloque de clustering, el interfaz IGroupCatalog define las operaciones que debe implementar un catálogo para ser compatible con el clustering. Este interfaz es heredado a su vez por IPersistentGroupCatalog e ITransientGroupCatalog, los cuales incluyen funcionalidades específicas de cada una de las variantes del catálogo consideradas. Por su parte, la clase CatalogGroup implementa las funciones de clustering.

- La clase BaseDatabase implementa las funcionalidades de más bajo nivel de la base de datos, mientras que IDatabase define las operaciones básicas de alto nivel de la misma. Las clases abstractas APersistentDatabase y ATransientDatabase implementan el interfaz IDatabase y extienden BaseDatabase, aunque en su código únicamente incluyen funciones específicas de bases de datos persistentes o transitorias, según sea el caso.

- El interfaz ICatalog define las operaciones que debe incluir una clase para ser considerada un catálogo. Este interfaz es implementado por dos clases abstractas, APersistentCatalog y ATransientCatalog, que a su vez hacen uso de PersistentDatabase y TransientDatabase, e implementan IPersistentGroupCatalog e ITransientGroupCatalog respectivamente.

- Si acudimos al último nivel del árbol de clases, podemos encontrar los servicios de catálogo ofrecidos, como son PersistentCatalog y TransientCatalog. Estas clases heredan de APersistentCatalog y ATransientCatalog, y hacen uso de las clases PersistentDB y TransientDB, que heredan de PersistentDatabase y TransientDatabase respectivamente.

La organización expuesta de interfaces y clases facilita la generación de nuevos catálogos, al separar la implementación en bloques lógicos. De esta forma para incluir un nuevo catálogo (al cual llamaremos Ejemplo) en el sistema, se deberían seguir los siguientes pasos:

- Definición del esquema de bases de datos en un script SQL. Este script se aloja dentro del propio paquete binario del servicio web.

- Implementación de la base de datos EjemploDB, heredando de PersistentDatabase o TransientDatabase, según se desee crear un catálogo persistente o transitorio, respectivamente.

    - La base de datos consiste en la implementación de las funciones de alto nivel de IDatabase, haciendo uso de las operaciones de bajo nivel de BaseDatabase. En la mayoría de los casos esta implementación es sencilla si se utilizan las funciones estándar de inserción, borrado, modificación y consulta, aunque se deja al desarrollador la libertad de elegir qué operaciones de alto nivel propias del catálogo “Ejemplo” quiere implementar.

- Implementación de la clase EjemploCatalog, que heredará de PersistentCatalog o TransientCatalog, según se desee un tipo de catálogo u otro. Esta clase hace uso de EjemploDB.

    - EjemploCatalog consiste en la implementación de las funciones de alto nivel de ICatalog, haciendo uso de las operaciones básicas de PersistentCatalog o TransientCatalog. Tal y como ocurre con la base de datos, la implementación es directa si se utilizan las funciones estándar de inserción, borrado, modificación y consulta, con la ventaja de que de esta forma se permite que el desarrollador defina nuevas funciones de alto nivel propias del catálogo concreto.

A continuación se explican detalles de la implementación de cada uno de los bloques lógicos, para el caso concreto de nuestra plataforma.

Front end Para el prototipo se ofrecen dos servicios de catálogo, que gestionan los datos persistentes de la plataforma (mediante el servicio PersistentCatalog) y los transitorios (mediante el servicio TransientCatalog).

PersistentCatalog es un catálogo persistente que almacena información sobre servicios, instancias y SLA, tal y como se puede ver en el esquema SQL de la Figura 4.5. Esta clase delega las operaciones básicas de inserción, borrado, modificación y consulta a APersistentCatalog (que incluye una implementación por defecto), e implementa la nueva función addService.

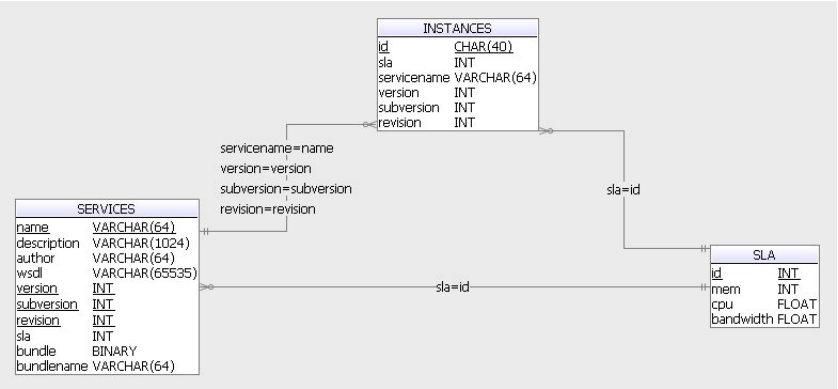

Figura 4.5: *Esquema SQL de la base de datos de PersistentCatalog.*

AddService es la función encargada de registrar nuevos servicios en la plataforma, y ha sido necesaria su creación porque las peculiaridades de esta

operación no encajan con las operaciones básicas ofrecidas por APersistentCatalog.

Para registrar un servicio en el catálogo, se debe insertar en la base de datos los metadatos del propio servicio, junto con una copia del paquete binario del mismo. En el catálogo persistente, las operaciones de inserción deben ser redirigidas a todos los nodos para mantener la coherencia de las bases de datos, y este es el comportamiento por defecto de la función "insert" incluida en la clase APersistentCatalog. Sin embargo, esta función está preparada para hacer broadcasting de mensajes cuyo contenido consta de información textual de tamaño reducido, debido a las limitaciones que impone JGroups. Por tanto no posee la funcionalidad necesaria para distribuir un fichero binario entre los nodos del grupo. De esta forma, para llevar a cabo la implementación de la función addService es necesario replicar el comportamiento de la función de inserción realizando la distribución de los metadatos del servicio y del paquete binario del mismo.

Por otro lado, TransientCatalog es un catálogo transitorio que almacena información sobre las instancias y los contenedores virtuales de la plataforma, tal y como se muestra en la Figura 4.6. Las funciones básicas de inserción, borrado, actualización y consulta han sido delegadas a "ATransientCatalog". Adicionalmente "TransientCatalog" incluye la función "removeInstances", que recibe como parámetro el identificador de un *Virtual Container* y procede a eliminar todas las instancias asociadas con el mismo. Esta función se utiliza cuando un *Virtual Container* falla, y por tanto su información debe ser eliminada de la base de datos.

Figura 4.6: *Esquema SQL de la base de datos de TransientCatalog.*

Clustering El bloque de clustering recoge las clases que implementan la estructura distribuida del catálogo.

Para ello, cada catálogo incluye una instancia de la clase CatalogGroup, que es la que implementa la pertenencia a grupos. Esta clase realiza la tarea análoga a la clase Group de la librería PaaSlib, generando un clúster entre catálogos del mismo tipo. Cada clúster de catálogos tiene un identificador único, que en el caso de los catálogos se ha hecho coincidir con el identificador de la base de datos. Utilizando este identificador, la clase CatalogGroup

conecta a la instancia del catálogo al grupo.

En el caso de nuestra implementación, para realizar la configuración de un grupo, la clase CatalogGroup utilizará un archivo de configuración .xml que deberá estar alojado en la carpeta conf del servicio con el mismo nombre del grupo (por ejemplo si un grupo se llama A, CatalogGroup busca el fichero conf/A.xml).

Bases de datos Todas las bases de datos disponen de unas operaciones de bajo nivel, como son inserción, borrado, modificación y consulta, que son las que realiza el SGBD de forma directa sobre el sistema. Para simplificar la generación de catálogos genéricos, estas operaciones se han agrupado en una clase denominada BaseDatabase. De esta forma, esta clase, utilizando el driver de la base de datos, implementa el código de las cuatro operaciones básicas al nivel más bajo, es decir, interactuando de forma directa con el SGDB utilizando sentencias SQL.

Sin embargo, no son estas operaciones de tan bajo nivel aquellas que deseamos exponer a los usuarios finales, sino unas operaciones de alto nivel que toman unos parámetros de entrada y se encargan de construir las sentencias SQL que finalmente se mandan a la base de datos. Estas operaciones de alto nivel que se desean exponer al exterior se han incluido en el interfaz IDatabase. La utilización de un interfaz para presentar unas operaciones más genéricas facilita la implementación de nuevos catálogos y ayuda a modularizar el diseño del componente. Lo que se consigue es una centralización y desacoplamiento del perfil de las funciones de alto nivel con respecto a su implementación.

La diferenciación entre bases de datos persistentes o transitorias se realiza en las clases PersistentDatabase y TransientDatabase, que heredan de BaseDatabase e implementan el interfaz IDatabase. Estas clases son las que realizan la conexión al driver de la base de datos subyacente y las que implementan las peculiaridades de cada variante de la misma.

La implementación de la lógica de la base de datos transitoria resulta sencilla, ya que consiste simplemente en generar una base de datos in-memory, utilizando las características de HSQLDB. Esta base de datos desaparece al terminar la ejecución del programa, sin dejar ficheros innecesarios en el disco. En contraposición a esto, la implementación de la base de datos persistente no resulta trivial, ya que conlleva la toma de decisiones de diseño sobre la lógica de la misma.

Una base de datos volátil puede generar un identificador único en cada ejecución. Sin embargo, una base de datos persistente los identificadores

deben perdurar entre ejecuciones, y para ello es necesario conocer el identificador de una base de datos existente a la que conectarse.

Cuando un catálogo persistente arranca, se une al clúster de catálogos. En caso de no ser el primer miembro del clúster, se une al grupo y deberá recibir el estado del sistema de uno de los nodos miembro. El proceso de recuperación de estado se realiza mediante las operaciones de JGroups (mediante callbacks a funciones de getState y setState), que, en la actual implementación, consiste en generar un backup de la base de datos y enviarla al nodo nuevo para que la restaure. En caso de ser el primer nodo del clúster, procede a recuperar el estado de la base de datos previa, o bien generar uno nuevo.

La información necesaria para llevar a cabo estas operaciones se suministra al sistema mediante una serie de parámetros, que son incluidos en el fichero de configuración XML del servicio. En el código x se muestra un ejemplo de configuración.

El parámetro "state" indica la forma en la que se va a recuperar el estado de la base de datos. En este caso, el valor "database" indica que se va a conectar a una base de datos existente identificada por el parámetro "dbname", o que se va a generar una nueva con ese identificador en caso de que no exista. Se ha implementado una alternativa, cuyo identificador es "backup", que indica que el estado se va a recuperar de una copia de seguridad desde el fichero indicado por "backupPath".

Como indicamos anteriormente, para generar una nueva base de datos, hace falta un esquema SQL, alojado en el paquete del propio servicio web, y un estado inicial. La motivación de este estado inicial es que en algunos casos podemos necesitar inicializar la base de datos con ciertos datos, como es el caso es la información sobre los *core services* de la plataforma. Para proporcionar esta información inicial, el parámetro "statePath" indica la ruta donde se encuentran estos datos.

La información básica de los *core services* está formada por el paquete del servicio (fichero .aar) y los metadatos del mismo. Dichos metadatos incluyen información como la descripción, la versión, etc. y han sido representados utilizado ficheros de propiedades, que almacenan dicha información sobre cada servicio.

En la ruta statePath se alojan los ficheros .aar y .properties que definen el paquete y los metadatos de cada servicio. Cuando el catálogo explora el contenido de la ruta statePath, lee cada fichero .properties, tomando como nombre del servicio el nombre del fichero, y su correspondiente fichero .aar, e introduce en la base de datos los metadatos y el paquete binario del servicio. El metadato WSDL, debido a su extensa longitud, ha sido extraído del fichero

```
<service name=‘‘PersistentCatalog’’ scope=‘‘application’’>
     <description>
        Persistent Catalog service.
     </description>
     <parameter name=‘‘ServiceClass’’>
        cloud.paas.catalog.persistent.PersistentCatalog
     </parameter>
     <parameter name=‘‘cloudname’’>
        mycloud
     </parameter>
     <parameter name=‘‘state’’>
        database
     </parameter>
     <parameter name=‘‘statePath’’>
        C:/temp/resources/
     </parameter>
     <parameter name=‘‘dbname’’>
        persistent_db1
     </parameter>
     ...
<service name=‘‘PersistentCatalog’’ scope=‘‘application’’>
     <description>
        Persistent Catalog service.
     </description>
     <parameter name=‘‘ServiceClass’’>
        cloud.paas.catalog.persistent.PersistentCatalog
     </parameter>
     <parameter name=‘‘cloudname’’>
        mycloud
     </parameter>
     <parameter name=‘‘state’’>
        backup
     </parameter>
     <parameter name=‘‘backupPath’’>
        C:/hsqldatabase8538652001525820056.read.backup.tar.gz
     </parameter>
     ...
```

Script 2: *Fragmento de dos ficheros de configuración de un PersistentCatalog, con las dos alternativas de recuperación de bases de datos.*

de propiedades a un fichero .wsdl, con lo que el catálogo buscará este fichero y lo introducirá en la base de datos, si se encuentra en la ruta de configuración.

4.2.3. Manager

El componente de administración de la plataforma se divide a su vez en varios módulos, cada uno encargado de gestionar un aspecto distinto de la gestión de la plataforma. A lo largo de este apartado se va a comentar la implementación de los módulos *Manager*, *SLA Manager* y *Load Balancer*, que se encargan básicamente de administrar los servicios, de gestionar los SLA y de equilibrar la carga del sistema, respectivamente.

Manager

El módulo "Manager" administra las operaciones generales de la plataforma, como son el alta y baja de servicios, y el despliegue, repliegue y actualización de instancias. Estas funciones se han implementado en un servicio web Java, que ofrece dichas funcionalidades al exterior mediante un interfaz público.

Estas funciones están limitadas a los roles de administrador y desarrollador, y por ello es necesario autentificar a los usuarios entrantes, utilizando el marco de seguridad incluido en PaaSlib.

La lógica de la administración de servicios e instancias es la siguiente:

- El identificador del servicio está formado por la tupla nombre, versión, subversión y revisión.
- Todo servicio debe incluir un SLA que define los términos de administración del mismo.
- El autor de un servicio es el usuario que registra por primera vez una versión del mismo en el sistema. En la versión actual de la plataforma, y con motivo de simplificar la administración del prototipo, el registro de subsiguientes versiones sólo podrá realizarlo el mismo autor.
- Del mismo modo, en esta versión del prototipo sólo el autor puede administrar un servicio. La administración de un servicio implica el poder realizar cualquiera de las operaciones disponibles sobre el módulo.
- Sólo se pueden desplegar instancias de servicios registrados.
- Las instancias pueden incluir o no un SLA propio, y en caso de que no se especifique, la instancia heredará el del servicio.

- La eliminación de un servicio implica el repliegue de todas sus instancias.
- Sólo se pueden actualizar instancias de una versión a otra registradas y, también para simplificar el desarrollo de la versión actual de la plataforma, la versión a la que se actualiza una instancia debe ser mayor a la inicial (siguiendo el orden versión.subversión.revisión).

Todas estas operaciones pueden implementarse utilizando SOAP estándar, a excepción de la operación mediante la que se realiza el registro de servicios. Para registrar un servicio, hay que proporcionar los metadatos, el SLA adjunto y el paquete binario (en este caso un fichero .aar) del mismo. El paquete del servicio es un fichero con un tamaño sensiblemente superior a los otros datos, y resulta potencialmente elevado. Frente a un caso con estas características no es posible el uso de SOAP estándar, y por ello hemos utilizado SOAP with Attachments [55] (SwA), que es una extensión que permite adjuntar ficheros a los mensajes SOAP mediante un mensaje "MIME multipart", de forma que se preservan las reglas de procesamiento SOAP. Axis2 incluye soporte para SwA, con lo cual es suficiente con habilitar esta funcionalidad en el contenedor para poder utilizarla en la programación de servicios, tanto por parte del cliente, como por parte del servidor.

SLA Manager

El módulo *SLA Manager* es el encargado de validar los SLA y de realizar operaciones relacionadas con dichos acuerdos de servicio. En el ámbito del prototipo, el *SLA Manager* sólo contiene dos de las operaciones especificadas en el diseño: validateServiceSLA y validateInstanceSLA. Cada una de estas operaciones se encarga de validar el SLA de servicios e instancias, respectivamente, devolviendo un valor booleano con el resultado de dicha operación.

Sin embargo, el resultado de la validación de un SLA puede ofrecer los siguientes resultados.

- El SLA no es válido. Las condiciones expuestas en el documento no cumplen con las políticas de la plataforma, violan los términos básicos de la misma (por ejemplo un desarrollador ha desplegado el número máximo de instancias permitidas) o el desarrollador no tiene crédito suficiente para soportarlas.
- El SLA es válido. Por defecto se asume que la plataforma puede cumplir un SLA válido.

- El SLA es válido, pero la plataforma no puede cumplirlo en ese momento. Un ejemplo sería solicitar desplegar una instancia en un contenedor con 512 MB de memoria, cuando no queda ninguno libre disponible. Técnicamente el SLA se ha validado satisfactoriamente, pero puesto que la plataforma no puede darle servicio, admitirlo sería equivalente a comenzar a soportar el SLA sin cumplirlo.

En nuestra implementación, para dar respuesta a estos 3 casos, el primero de ellos se ha implementado como un valor de retorno false, el segundo como un valor de retorno true y el tercer caso se resuelve mediante el lanzamiento de una excepción. De esta forma, los tres casos se pueden diferenciar claramente y ser tratados de forma adecuada por funciones externas.

Para simplificar la implementación del prototipo, el proceso de validación se limita a comprobar que el usuario tiene permisos para registrar servicios, y que existen contenedores libres capaces de albergar el despliegue de una nueva instancia.

Load Balancer

El módulo *Load Balancer* se encarga de equilibrar la carga del sistema distribuyendo peticiones entre los elementos disponibles (*Virtual Containers*). En el ámbito del prototipo, se ha implementado la función getTargetVirtualContainer, que se encarga de elegir el contenedor más adecuado donde se debe desplegar la instancia de un servicio. Para ello se utiliza el SLA de los contenedores y la información adicional disponible sobre ellos.

En el prototipo la función de equilibrado de carga se limita a elegir aleatoriamente un contenedor como objetivo, eliminando de la lista aquellos que ya han desplegado una instancia del servicio solicitado.

4.2.4. Virtual Container

El *Virtual Container* es el *runtime* virtualizado, donde se alojan y ejecutan las instancias de los servicios de la plataforma. Puesto que el tipo de servicios virtualizados en el prototipo son Servicios Web Java, se ha utilizado Apache Axis2 como base para este componente, tal como se adelantó en el apartado4.2.1.

Sin embargo un *Virtual Container* no consta únicamente del *runtime* de los servicios, sino que este debe ser instrumentado y adaptado a las características del sistema para poder realmente ofrecer un entorno virtualizado para la plataforma.

En el ámbito del Prototipo no se ha instrumentado el contenedor, puesto que este proceso implica modificar el código fuente del mismo para añadir funcionalidades para este componente (por ejemplo la traza de estadísticas o métricas de la plataforma). Sin embargo, la relajación de ciertas características para crear el prototipo ha hecho que no sea necesario realizar las modificaciones, por el momento, a pesar de que se han determinado algunas zonas del código que deben ser modificadas para implementar las características adicionales necesarias.

De esta forma, se ha realizado una implementación de un core service "VirtualContainer", que incluye las operaciones de administración de servicios a nivel de contenedor, como son desplegar, replegar y actualizar instancias. Adicionalmente se ha incluido un módulo *Reporter*, que se encarga de enviar informes al catálogo sobre las actividades que se llevan a cabo en el contenedor, como son el despliegue o repliegue de instancias.

El componente *Virtual Container*, por tanto, está conformado por un contenedor Apache Axis2, acompañado por todas sus librerías y ficheros de configuración, junto con un servicio web que permite acceder a sus funcionalidades desde el exterior. Para implementar alguna de las funcionalidades de administración de servicios, el módulo VirtualContainer accede directamente a las clases internas de Axis2, que permiten realizar estas operaciones programáticamente.

En cuanto al funcionamiento con respecto a la plataforma, para desplegar una instancia, tras comprobar que en el *Virtual Container* no se encuentra ya desplegada una instancia del servicio, el módulo consulta los catálogos y recupera los metadatos de dicho servicio. Junto con los metadatos, el *Virtual Container* descarga el paquete del servicio, alojándolo en la ruta especificada por la configuración de Axis2. El proceso de despliegue se realiza programáticamente mediante la utilización del *runtime* de Axis2. Para ello, es necesario cargar las clases contenidas en el paquete en el Classloader, y explorar el contenido del paquete para extraer el fichero de configuración del servicio. Utilizando el fichero de configuración y la ruta del paquete, la clase DeploymentEngine introduce los datos del servicio en una instancia de la clase AxisService mediante el método buildService. La instancia de AxisService con la información del servicio introducida puede entonces ser desplegada en el contenedor mediante el método deployService de la clase ConfigurationContext.

El proceso de repliegue resulta mucho más sencillo que el de despliegue ya que para replegar una instancia, el contenedor simplemente comprueba que efectivamente la instancia señalada se encuentra desplegada en el contenedor, y si es así, para y elimina el servicio del contenedor mediante los

métodos "stopService" y "removeService" de la clase AxisConfiguration. Posteriormente elimina las referencias a la instancia del catálogo y finalmente borra del sistema de ficheros el paquete binario del servicio.

El módulo "Reporter" se encarga de registrar en los catálogos transitorios la información sobre las instancias que se despliegan y repliegan en cada contenedor mediante una suscripción por método *Push*. Para ello, toma "k" de estos catálogos de forma aleatoria (siendo k configurable) y les manda informes cuando se producen estos eventos. Este módulo debe manejar las situaciones singulares como aquellas en las que uno de los catálogos que maneja ha dejado de funcionar o el caso en que catálogos nuevos se añaden a la plataforma.

A partir de aquí se puede dar la siguiente casuística con respecto a los nodos que forman los catálogos:

- Si uno de los catálogos a los que se envían informes ha dejado de funcionar, el módulo escoge de forma aleatoria uno nuevo.
- En caso de que se añadan catálogos nuevos al sistema, el módulo escoge aleatoriamente un nuevo conjunto de k catálogos de forma aleatoria, con el objetivo de redistribuir la carga entre los mismos. Si se van a elegir nuevos catálogos, el *Virtual Container* ya no va a estar suscrito a los antiguos, y por tanto no va a actualizar la información que haya almacenado en ellos. Para evitar incoherencias y mantenimiento de información no actualizada en el sistema, el *Virtual Container* mandar orden a los catálogos viejos de borrar toda la información almacenada concerniente a él (por ejemplo las instancias que tiene desplegadas).

4.2.5. User Interface

El interfaz de usuario está compuesto por la parte cliente de los *core services* de la plataforma que ofrecen funcionalidad al exterior. En el caso del prototipo que se describe en esta memoria, estos componentes son Manager, TransientCatalog y PersistentCatalog. Puesto que estos tres componentes han sido implementados como servicios web Java para el contenedor Axis2, se han utilizado las herramientas que proporciona el propio Axis2 para generar la parte cliente de los propios servicios.

Axis2 incluye una herramienta denominada Codegen, que permite generar código automáticamente. Mediante el comando wsdl2java, Codegen genera el código cliente (stub) a partir del documento WSDL de un servicio. El código generado automáticamente incluye las excepciones y las clases no primitivas utilizadas en los parámetros de las funciones, y una clase Stub

que implementa un proxy a las funciones del servicio. El stub posee el mismo interfaz que el servicio al que representa, y los argumentos del constructor son un ConfigurationContext (contexto de Axis2) y la Endpoint reference del servicio al cual se desea conectar.

Para conectar a un servicio utilizando un stub, es suficiente con instanciar una clase del mismo utilizando un ConfigurationContext cargado con la configuración del cliente (un fichero de configuración XML) y la referencia al servicio destino. A partir de ahí se pueden utilizar sus funciones a modo de RPC. El stub se encarga de enmascarar, mediante sus funciones, la construcción del mensaje SOAP, el envío al destino y la recepción de la respuesta, realizando la comunicación de forma transparente.

Para el Prototipo, se han generado los stubs de los tres *core services* públicos, y se han utilizado para implementar el cliente en forma de páginas web basadas en JSP. A partir de estas páginas web, mediante la utilización de formularios, los clientes pueden realizar las operaciones de administración de servicios o realizar consultas a los catálogos de una forma muy visual y sencilla.

4.3. Despliegue del prototipo

El prototipo se compone de cuatro servicios web Java que implementan los módulos de Manager, PersistentCatalog, TransientCatalog y VirtualContainer, además de un contenedor de servicios Apache Axis2 que actúa como *runtime* de los mismos.

Para poner en funcionamiento los módulos de la plataforma, es necesario alojar los paquetes binarios del servicio web en la ruta repository/services/ de Axis2. Un paquete binario de un servicio web Java para Axis2 es un fichero comprimido con el algoritmo PKZIP con extensión .aar que contiene los ficheros binarios, las librerías y el fichero de configuración XML del servicio web. La distribución de Cloud ComPaaS proporciona una serie de scripts que permiten generar de forma sencilla los paquetes binarios de la plataforma a partir de los ficheros que se deben incluir en los mismos. Estos scripts (proporcionados para Windows y Linux) se encargan de comprimir el contenido de la carpeta bin y las carpetas META-INF, lib y conf de cada core service, y de generar un fichero comprimido, con extensión .aar.

Para realizar el despliegue de alguno de los componentes de la plataforma, basta con alojar los paquetes binarios en el repositorio de un contenedor Axis2 y poner en marcha el mismo. Al iniciar su ejecución, los servicios se incorporan al clúster de la plataforma y tras las labores de inicialización,

estarán disponibles para su utilización. Cabe señalar a este respecto que, en el prototipo actual, el servicio "VirtualContainer" realiza un chequeo de los servicios desplegados en el contenedor en su inicialización, y procede a eliminar del mismo a todos aquellos que no sean *core services*. El objetivo de esta operación es el de proporcionar un despliegue "limpio" del contenedor a la plataforma.

Para habilitar las funciones de clustering de la plataforma, Cloud ComPaaS utiliza ficheros de configuración JGroups. Estos ficheros deben ser alojados en el directorio conf/ bajo el nombre jgroups.xml, y definen la configuración de todos los aspectos del clustering. En nuestro caso, se han definido tres grupos, dos de catálogos y uno general. Los dos grupos de catálogos se corresponden con las dos variantes del catálogo, y agrupan a los catálogos persistentes y transitorios respectivamente. El grupo general de la plataforma es el que conforma el clúster compuesto por todos los nodos del despliegue, y su cometido es el de permitir la comunicación interna de los nodos.

A la hora de desplegar una nueva plataforma Cloud ComPaaS, el aspecto de configuración más importante es la especificación de los "hosts iniciales". Estos "hosts iniciales" son aquellos nodos de la plataforma que van a actuar como coordinadores del grupo general, comunicándose con los nuevos nodos que desean incorporarse al mismo. Cuando un nuevo nodo comienza su ejecución, procede a conectarse a alguno de los nodos iniciales con el fin de obtener la información del grupo y e incorporarse al mismo. En caso de que no sea posible conectarse a dichos nodos, o el módulo que arranca es uno de los nodos iniciales, JGroups genera un nuevo grupo de ejecución con dicho nodo.

La información sobre la dirección de los nodos iniciales se proporciona en el fichero JGroups. Por ello, es necesario que estos nodos tengan direcciones bien conocidas y accesibles públicamente. Esta condición es asumible puesto que son los nodos que forman la infraestructura del sistema. Adicionalmente es conveniente que estos sean los primeros nodos en comenzar a ejecutarse del grupo, ya que en otro caso podrían producirse comportamientos incorrectos en el sistema.

Este mismo procedimiento se aplica a los grupos de catálogos, pues funcionan exactamente de la misma manera. Para cada uno de los catálogos presentes en el sistema (dos en el prototipo), se utiliza un fichero de configuración JGroups en el que se especifica la información sobre el grupo al que deberán pertenecer dichos catálogos. Este fichero se aloja en la ruta config/ del paquete del servicio, y consiste en un fichero con el nombre del identificador del catálogo (por ejemplo si el catálogo tiene como identificador A, el fichero se encuentra en la ruta config/A.xml).

A pesar de que los servicios web pueden ser alojados en cualquier contenedor Axis2, y otros contenedores de servicios web Java, la distribución que proporcionamos incluye un contenedor Axis2 completo. Este contenedor es el que ha sido utilizado para las labores de testeo del prototipo, e incluye todas las librerías y ficheros necesarios para ejecutar los servicios. Para nuestro propósito hemos tenido que realizar las siguientes modificaciones en la configuración de la distribución estándar:

- Modificación de la configuración de Axis2 para soportar SwA.
- Inclusión de las librerías de Apache Rampart.
- Inclusión de almacenes de claves y certificados. Estos ficheros son utilizados por la plataforma como credenciales, con el objetivo de securizar el procedimiento de incorporación a la plataforma.

De estos tres elementos, merecen especial atención las credenciales de la plataforma. La distribución del piloto incluye unos ficheros de credenciales de prueba. Sin embargo, es conveniente sustituir dichos ficheros por unas credenciales generadas por el administrador, con el objetivo de garantizar la seguridad del sistema.

La parte cliente de los servicios se ha distribuido, para cada uno de los servicios que proporcionan funcionalidad al exterior, en concreto Manager, PersistentCatalog y TransientCatalog, en dos formatos, por un lado, como una aplicación de línea de comandos y, por otro, como páginas web JSP. Estos clientes realizan llamadas SOAP a las funciones del interfaz público de los servicios, recuperan la respuesta y la muestran al usuario. Para ello, necesitan la Endpoint Reference del servicio al cual se van a conectar, así como los parámetros necesarios para cada una de las funciones que realizan.

Capítulo 5

Producción científica

En este capítulo se cita el artículo publicado en el congreso Cracow Grid Workshop '09 en el que se realiza la descripción de la arquitectura básica de Cloud ComPaaS y una descripción resumida de los módulos que la componen.

1. Andrés García, Carlos de Alfonso, and Vicente Hernández. Design of a Platform of Virtual Service Containers for Service Oriented Cloud Computing. In Cracow Grid Workshop '09 Proceedings, pages 2027, Cracow, Poland, 2010. ACC CYFRONET AGH.

En la actualidad se está preparando un artículo para la revista Future Generation Computer Systems (FGCS), en el que se describe el sistema de catálogos de la plataforma. En el artículo se plantea la necesidad de un Sistema de Información integrado para las soluciones Cloud PaaS, se analizan los desarrollos existentes y se presenta el diseño del catálogo de Cloud ComPaaS. A continuación se detalla el funcionamiento de las variantes de catálogo y se muestra cómo su diseño se ajusta a las necesidades de un Sistema de Información Cloud, específicamente orientado a soluciones PaaS, y cómo el catálogo de Cloud ComPaaS proporciona otras características deseables para la plataforma.

De forma adicional se ha puesto en marcha una página web asociada a Cloud ComPaaS, alojada en http://www.grycap.upv.es/compaas. En esta web se publica la información relativa al proyecto, y que ofrece una distribución del mismo y documentación para su utilización. El objetivo de esta web es proporcionar a los usuarios una forma de acceder al producto, y crear una comunidad que por un lado se beneficie de la utilización de Cloud ComPaaS y por otro lado realice aportaciones al proyecto, bien sea en forma de comentarios, sugerencias o colaboraciones.

Capítulo 6

Conclusiones y trabajos futuros

Para el desarrollo de la tesis de master, en primer lugar se ha hecho la labor de búsqueda de un campo actual aún poco explorado, donde iniciar una línea de investigación y, a partir de ahí, se ha abordado una tarea de recopilación y procesado de documentación acerca de las tecnologías involucradas. Este trabajo ha llevado a realizar una especificación de la plataforma desde un punto de vista científico, procurando prescindir de la búsqueda de resultados comerciales, para así tratar el problema desde un punto de vista más formal. Este método de trabajo se ha aplicado como contraposición a las urgencias que tienen las soluciones comerciales en un campo emergente, que buscan obtener un tipo de producto que cubra las expectativas de los potenciales consumidores, sin atender a las necesidades futuras o las características más científicas.

A partir de esta especificación se ha procedido a realizar el diseño de una arquitectura que da soporte a la plataforma y a definir una serie de los módulos que permiten crear una plataforma conforme a dicha arquitectura. Posteriormente se ha realizado una implementación de un prototipo que ha demostrado la validez y corrección de los principios fundamentales de la arquitectura.

En cuanto al diseño y la implementación, gracias a una organización modular y desacoplada, se facilita la división de la plataforma en bloques lógicos, y esto permite modificar, mejorar y sustituir componentes sin afectar al funcionamiento del sistema. Esta aproximación facilita también la adaptación de la herramienta a distintos entornos y situaciones, y permite realizar desarrollos en sectores concretos de la misma.

Podemos considerar que los objetivos de la Tesis de Máster han sido cubiertos a lo largo de su desarrollo, llegando a generar como resultado Cloud

ComPaaS, que es una herramienta para la creación de nubes PaaS privadas, lista para distribuir en ámbitos científicos, y que supone un hito en cuanto al campo de Cloud Computing PaaS.

La herramienta descrita permite a las organizaciones poner en marcha nubes PaaS privadas, utilizando sus propios recursos computacionales. Se ha optado por utilizar una aproximación para la distribución de la plataforma en forma de software libre y código abierto, con el objetivo de que las organizaciones puedan modificar la herramienta para adaptarla a sus propias necesidades. Esto permitirá a los potenciales usuarios aprovechar los beneficios del Cloud Computing, sin la necesidad de acudir a un proveedor externo. Esta decisión se refuerza por el hecho de que en la actualidad las soluciones comerciales para este tipo de plataformas, ofrecen características dispares, ya que cada una de ellas oferta soporte para distintos tipos de lenguajes, estándares o funcionalidades. Por otro lado existe un con escaso consenso en cuanto a lo que sería la definición de las características de una nube PaaS, con respecto a estas plataformas comerciales.

Por otro lado, el espíritu de creación de Cloud ComPaaS es que no solamente trate de ser una herramienta para la adopción del Cloud PaaS por parte de diversas organizaciones, sino que se pretende que sirva como banco de pruebas para desarrollos científicos en áreas relacionadas con el Cloud Computing.

Desde el punto de vista científico, Cloud ComPaaS abarca un amplio abanico de conceptos relacionados con el Cloud Computing, algunos de los cuales conforman líneas de investigación en sí mismos. Y, si bien el objetivo del prototipo actual no es la profundización en todos y cada uno de estos aspectos, la herramienta permite la especificación, desarrollo, implementación y testeo de distintos estándares, protocolos o algoritmos. A continuación se citan algunas de las características de la plataforma que conforman problemas abiertos, líneas de investigación y trabajos futuros que se plantean con Cloud ComPaaS.

- Migración en vivo: La aplicación de la migración en vivo de los servicios virtuales requiere del desarrollo de un protocolo de migración, así como la definición de ciertos aspectos como la redirección de usuarios al elemento migrado o qué hacer con los clientes activos de un elemento que se va a migrar (terminar su sesión, dejarles continuar hasta terminar, etc.).

- Interoperabilidad: Es necesario definir un lenguaje para expresar peticiones y concesiones de recursos, un protocolo de negociación entre

nubes y un algoritmo para la cesión de recursos entre dos nubes, entre otros aspectos. Adicionalmente se deben definir mecanismos de pago o de evaluación de nubes, que permitan discernir la mejor opción a utilizar en cada negociación.

- SLA: Es necesario definir con más detalle los términos de los distintos SLA de la plataforma, así como la forma en la cual estos son transmitidos, almacenados y procesados. Es necesario completar la forma en que estos SLA interactúan con distintos elementos de la plataforma, así como los mecanismos que permiten comprobar un SLA y determinar si es válida o si se está cumpliendo en el instante actual.

- Calidad de Servicio: Para poder realizar un control de la calidad de servicio en una nube PaaS es necesario definir indicadores de calidad de servicio significativos para la plataforma, así como la forma en la cual estos son recopilados y analizados. Algunos ejemplos serían el uso de CPU, uso de memoria, número de clientes activos para cada instancia, etc., que requerirían, entre otras cosas, el soporte por parte del *runtime* (en nuestro prototipo, Axis2).

- Almacenamiento Cloud: Es necesario adoptar interfaces de almacenamiento Cloud estándares, como CDMI [56], así como la forma en la cual se va a implementar el soporte a dicho almacenamiento. La implementación debe ofrecer, entre otras características, distribución y replicación de datos de forma transparente al usuario.

- Seguridad: Resulta conveniente ampliar el marco de seguridad de la plataforma con nuevos esquemas de autentificación y autorización (por ejemplo, los basados en aserciones SAML y autenticación mediante certificados X.509), así como proporcionar métodos de delegación de credenciales.

- Elasticidad: Es necesario definir un lenguaje para expresar reglas de elasticidad. Probablemente estas reglas deberán ser expresadas en términos de indicadores de calidad de servicio, con lo que este módulo está supeditado a la Calidad de Servicio. Otra área en la que sería conveniente trabajar es la efectividad de las reglas de elasticidad sobre la mejora de los parámetros de calidad de servicio, la forma más eficiente de definir tales reglas, la posibilidad de definir reglas adaptativas, etc.

- Auto recuperación: Es necesario definir cómo la plataforma detecta elementos fallidos y cómo actúa en presencia de una partición del clúster

de la plataforma.

En definitiva, el estudio del campo PaaS dentro del Cloud Computing abre numerosas líneas de investigación, debido principalmente al hecho de que se han invertido la mayoría de recursos en el desarrollo de la capa IaaS.

Este trabajo supone la primera parte del proyecto de Tesis Doctoral, en el que se va a continuar desarrollando Cloud ComPaaS, profundizando en el diseño de los distintos componentes, y aumentando el alcance del prototipo hasta obtener una versión completa de la plataforma, que incluya todos los módulos y dé soporte a todas las funcionalidades de la misma.

Bibliografía

[1] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[2] Amazon.com Inc. Amazon EC2 (Elastic Compute Cloud) , 2010. [Online; accessed 17-09-2010].

[3] Google Inc. Google App Engine , 2010. [Online; accessed 17-09-2010].

[4] Salesforce.com Inc. . Salesforce.com , 2010. [Online; accessed 17-09-2010].

[5] Amazon.com Inc. Amazon Simple Storage Service (S3) , 2010. [Online; accessed 17-09-2010].

[6] Nirvanix Inc. Nirvanix , 2010. [Online; accessed 17-09-2010].

[7] Savvis Inc. Savvis , 2010. [Online; accessed 17-09-2010].

[8] GoGrid Cloud Hosting . GoGrid , 2010. [Online; accessed 17-09-2010].

[9] 3tera Inc. 3Tera , 2010. [Online; accessed 17-09-2010].

[10] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *Proceedings of Cloud Computing and Its Applications*, October 2008.

[11] Universidad de Chicago . Nimbus , 2010. [Online; accessed 17-09-2010].

[12] OpenNebula Project Leads . Open Nebula , 2010. [Online; accessed 17-09-2010].

[13] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio M. Llorente, Ruben Montero, Yaron Wolfsthal, Erik Elmroth, Juan Caceres, Muli Ben-Yehuda, Wolfgan Emmerich, and Fermin Galan. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development*, 53(4), 2009.

[14] John Markoff. Internet critic takes on microsoft. *The New York Times*, April 9, 2001.

[15] Eric Schmidt. Conversation with eric schmidt hosted by danny sullivan , search engine strategies conference, August 9, 2006.

[16] Bits contributors. Why can't we compute in the cloud? *The New York Times*, August 24, 2007.

[17] Princeton University's Center for Information Technology Policy. Computing in the cloud workshop, Januray 14-15, 2008.

[18] eBay Inc. eBay , 2010. [Online; accessed 17-09-2010].

[19] Google Inc. Google Docs , 2010. [Online; accessed 17-09-2010].

[20] David Chappell. A short introduction to Cloud Platforms: An enterprise-oriented view. Technical report, Chappell & Associates, August 2008.

[21] Larry Ellison. Quote from larry ellison, September 26, 2008.

[22] Gartner Inc. Gartner's 2009 Hype Cycle Special Report Evaluates Maturity of 1,650 Technologies . Technical report, Gartner Inc., August 11, 2009.

[23] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, February 2009.

[24] Engine Yard Inc. Engine Yard Cloud , 2010. [Online; accessed 17-09-2010].

[25] RightScale Inc. RightScale , 2010. [Online; accessed 17-09-2010].

[26] Microsoft Corporation . Windows Azure , 2010. [Online; accessed 17-09-2010].

[27] Sun Microsystem . Network.com, posteriormente renombrado Sun Cloud , 2010. [Offline. Sun Cloud fue cancelado tras la adquisición de Sun Microsystem por parte de Oracle Corporation].

[28] International Business Machines Corp. . Blue Cloud , 2010. [Online; accessed 17-09-2010].

[29] Abiquo . Abicloud , 2010. [Online; accessed 17-09-2010].

[30] James Maguire, Jeff Vance, and Cynthia Harvey. 85 Cloud Computing Vendors Shaping the Emerging Cloud . Technical report, QuinStreet Inc., August 25, 2009.

[31] Gartner Inc. Gartner Says Worldwide Cloud Services Revenue Will Grow 21.3 Percent in 2009 . Technical report, Gartner Inc., August 11, 2009.

[32] Michelle Bailey, Matthew Eastwood, Al Gillen, and Dhaval Gupta. Server Virtualization Market Forecast and Analysis 2005-2010 . Technical report, IDC, 2005.

[33] VMWare Inc. . VMware Virtual Platform , 2010. [Online; accessed 17-09-2010].

[34] Intel Corporation . Intel-VT , 2010. [Online; accessed 17-09-2010].

[35] Advanced Micro Devices, Inc. . AMD-V , 2010. [Online; accessed 17-09-2010].

[36] W3 Consortium . Web Services Architecture , 2010. [Online; accessed 17-09-2010].

[37] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language . Technical report, W3C, 2007.

[38] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP Version 1.2 . Technical report, W3C, 2007.

[39] Oracle Corporation. Oracle Service-Oriented Architecture , 2010. [Online; accessed 17-09-2010].

[40] International Business Machines Corp. IBM Service Oriented Architecture (SOA), 2010. [Online; accessed 17-09-2010].

[41] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[42] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.

[43] EGEE. Enabling Grids for E-sciencE, 2010. [Online; accessed 06-04-2010].

[44] L. Joita, Omer F. Rana, Felix Freitag, Isaac Chao, Pablo Chacin, Leandro Navarro, and Oscar Ardaiz. A catallactic market for data mining services. *Future Gener. Comput. Syst.*, 23(1):146–153, 2007.

[45] Domenico Talia and Paolo Trunfio. Web services for peer-to-peer resource discovery on the grid. In *In Delos Workshop: Digital Library Architectures*, pages 24–25, 2004.

[46] SNIA. SNIA Cloud Storage Initiative, 2010. [Online; accessed 17-09-2010].

[47] The Apache Software Foundation. Apache Axis2, 2010. [Online; accessed 17-09-2010].

[48] Sun Microsystems. Java Servlet Technology . Technical report, Sun Microsystems, 2010.

[49] The Apache Software Foundation . Apache License 2.0, 2010. [Online; accessed 06-04-2010].

[50] The Apache Software Foundation. Apache Rampart, 2010. [Online; accessed 17-09-2010].

[51] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. WS-SecurityPolicy 1.2 . Technical report, OASIS, 2007.

[52] The hsql Development Group. HyperSQL DataBase, 2010. [Online; accessed 17-09-2010].

[53] PolePosition. PolePosition, 2010. [Online; accessed 17-09-2010].

[54] Bela Ban. JavaGroups - Group Communication Patterns in Java. Technical report, 1998.

[55] John J. Barton, Satish Thatte, and Henrik Frystyk Nielsen. SOAP Messages with Attachments . Technical report, W3C, 2000.

[56] SNIA. SNIA Cloud Data Management Interface, 2010. [Online; accessed 17-09-2010].