Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Desarrollo de un modelo estándar para los servicios ofrecidos por las interfaces de programación de aplicaciones (APIs) usadas en cómputo en la nube

Tesis presentada como requisito para obtener el título de Maestría en Ciencias de la Computación

Octubre 2012

Presenta

Miguel Felipe Pérez Escalera

Asesores

Dr. Luis Carlos Altamirano Robles **Dr. Miguel Ángel León Chávez**

Le dedico este trabajo de tesis a quienes, de una u otra forma, siempre están conmigo.

A América, mi mamá: Que siempre has creído en mí y tu apoyo e incondicional amor no me han faltado un solo día.
A Adrián, mi papá: Porque siempre te has interesado en mis proyectos y estás conmigo en los momentos más complicados.
A Toni, mi hermana: Que gracias a tu amor, compañía y frescura no me volví loco haciendo la tesis y me la pasé muy feliz.
A mis amigos: Qué no importa si me aparto por un tiempo, cuando los vuelvo a ver me reciben con el mismo cariño… después de insultarme un poco, claro.

Agradezco a mis asesores Miguel Ángel y Luis Carlos, junto con todos mis profesores, que me mostraron lo gratificate de compartir conocimiento y lo importante de contribuir a la ciencia desinteresadamente.

Y también le agradezco a quien sea que esté leyendo esta tesis. Porque gracias a ti, se hace importante mi trabajo.

"Con cierta parte de nuestro ser vivimos todos fuera del tiempo"
Milan Kundera, "La inmortalidad"
1989

ATTE.
Miguel Escalera
migue0583@hotmail.com

Resumen rápido para el lector

Este trabajo propone el modelo de una Interfaz de Programación de Aplicaciones (API) para los servicios comunes usados en cómputo en la nube. Ataca el problema de la dependencia de una aplicación con la plataforma sobre la que fue desarrollada (lock-in) y busca solucionarlo al homogenizar el desarrollo en las nubes facilitando la portabilidad. El modelo incluye el análisis, diseño e implementación siguiendo el Proceso Unificado de Desarrollo de Software y utilizando el Lenguaje Unificado de Modelado (UML).

TABLA DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

1 INTRODUCCIÓN

El cómputo en la nube es una tendencia actual de desarrollo de aplicaciones y servicios Web mediante el uso de Internet. Permite a los desarrolladores el acceso a recursos computacionales aparentemente ilimitados con esquemas de pago muy flexibles. Se basa en la idea de utilizar la infraestructura de un proveedor de nube en vez de forzar al desarrollador a adquirir su propia infraestructura, como hardware y software.

Así como existen diversos sistemas operativos en el mercado, también hay múltiples proveedores de nube actualmente. Cada uno de ellos ha creado su nube de la forma que más le conviene de acuerdo a su modelo de negocio y, al igual que con los sistemas operativos, existen diferencias e incompatibilidades entre la infraestructura de cada nube.

Cada proveedor brinda una Interfaz de Programación de Aplicaciones (API) que contiene las clases y funciones necesarias para explotar las características de su nube particular. Estas APIs son distintas para cada nube debido a diferencia de conceptualización e infraestructura y negocio. Cuando el programador hace uso de ésta en sus aplicaciones, las hace dependientes de la nube para la cual las desarrolló. A este fenómeno se le llama *lock-in* y es indeseable cuando el desarrollador opta por migrar su aplicación de una nube a otra y se encuentra con que tiene que hacer un ajuste extenso a su código. No hay portabilidad de aplicaciones en la nube.

Para evitar la incompatibilidad entre nubes, algunos estándares han sido propuestos por diferentes entidades, tal como el NIST en su Definición de Cómputo en la Nube, NIST Sinopsis y recomendaciones para Cómputo en la Nube, IEEE P2301 Perfiles de Nube, IEEE P2302 Intercloud, ITU- T Cómputo en la nube y estandarización: FG Reportes técnicos de la nube, DMTF Especificación del formato abierto de virtualización (OVF), DMTF La

incubadora de estándares de nubes abiertas y el Open Cloud Manifiesto. Ninguno ha sido tomado como estándar internacional.

Para atacar el problema, en este trabajo de tesis se desarrolla el modelo de un API estándar abstracta que contiene los servicios comunes de las nubes y define la forma que podrían tener las bibliotecas de los proveedores para que el usuario acceda a sus recursos de una forma genérica ajena a la plataforma sobre la que está trabajando.

El API estándar se desarrolla utilizando el proceso unificado de desarrollo de software (PUDS) y el lenguaje unificado de modelado (UML). Se presentan los modelos de análisis, diseño, implementación y pruebas.

El modelo de análisis consta de los diagramas de casos de uso y clases; el modelo de diseño comprende varios diagramas entre ellos clases refinada, secuencia, colaboración, estados y actividades. El modelo de implementación consta de un prototipo del API estándar para dos nubes particulares; el modelo de pruebas valida el prototipo para un servicio en las nubes.

Finalmente se evalúa la calidad del API estándar, enumerando los parámetros de calidad y midiéndolos en el prototipo.

1.1 PROBLEMÁTICA

En este punto se discute el problema de portabilidad de aplicaciones entre distintas nubes y que se está intentando resolver en este trabajo. Para analizarlo es necesario revisar el concepto de portabilidad de software en general.

1.1.1 PORTABILIDAD DE SOFTWARE

La portabilidad de software se refiere a la capacidad que tiene un sistema de software para ser ejecutado en diferentes plataformas (hardware, sistema operativo y librerías) reutilizando su código, haciendo cambios mínimos o ninguno.

Cuando un sistema se codifica específicamente para una plataforma tendrá una portabilidad baja debido a su dependencia, mientras que si usa funcionalidad genérica entre diferentes plataformas su portabilidad será alta. El caso ideal es que un mismo código pueda ser transportado a una plataforma distinta de la que se usó para su creación y funcione sin ajustes.

Es importante notar que se está hablado a nivel de código fuente y no programas binarios. En el caso de los binarios, éstos son compilados para funcionar bajo una plataforma específica (hardware y sistema operativo) y no pueden ser transportados. Algunas

excepciones, como Java, generan un programa pseudo-compilado (byte code) que será interpretado por la Máquina Virtual de Java. Ésta funciona como una plataforma única independiente del sistema operativo que resuelve el problema de portabilidad de los binarios. Pero en el caso de código fuente nos referimos a que el mismo código antes de ser compilado pueda ser cambiado de ambiente y seguir teniendo acceso a las funciones y bibliotecas para las que fue inicialmente preparado y las maneje de la misma forma.

Lo que se discute en este trabajo es el problema de portabilidad que tiene el código fuente de las aplicaciones que se crean para una nube en particular y que no puede ser directamente transportado a otras debido a su dependencia. Este problema se presenta cuando una aplicación accede a las características específicas de la nube en la que se encuentra mediante el kit de desarrollo (el API) brindado por ésta y al cambiar de nube, intenta utilizar un kit no compatible con su código.

1.2 OBJETIVOS

- Objetivo general
    - Desarrollar el modelo de los servicios básicos utilizados por las aplicaciones de cómputo en la nube para proponer un estándar en la implementación de APIs.
- Objetivos Específicos
    - Hacer el análisis de las interfaces de desarrollo de aplicaciones de los proveedores de cómputo en la nube más populares.
    - Identificar la funcionalidad común.
    - Construir el modelo de análisis y diseño de una API abstracta estándar detallando sólo el servicio de almacenamiento.
    - Construir un prototipo del servicio de almacenamiento persistente.
    - Realizar una prueba del API estándar creada.
    - Evaluar la calidad del API estándar.

1.3 METODOLOGÍA

Ya que lo que se busca crear con este proyecto es un modelo para implementación de software, se hará uso de la metodología propuesta por el Proceso Unificado de Desarrollo de Software (PUDS) creado por Ivar Jacobson, Grady Booch y James Rumbaugh. En el que se describe el procedimiento para la planeación, desarrollo e implementación de software usando el Lenguaje Unificado de Modelado.

1.3.1 LA NECESIDAD DEL MODELADO

Cuando construimos software nos enfrentamos a un problema elemental: comprender el problema que va a resolver nuestro software. Y de acuerdo al tamaño del problema puede que la solución sea pequeña, si el problema es simple, o muy grande, si es complejo.

En el caso de los problemas simples, es común que se aborde la construcción de la solución en software sin mucha planeación puesto que es fácil ver el panorama completo e idear el sistema que lo soluciona. Incluso puede que sea una sola persona la que trabaje sobre la solución.

Pero cuando los problemas son más complejos, nos enfrentamos a un obstáculo, puesto que nuestras capacidades son limitadas y tener presentes todos los casos posibles: todas las entradas, todas las salidas, todos los puntos potenciales de riesgo, etc. es casi imposible; más cuando la construcción del software se debe realizar por un grupo grande de personas coordinadas.

Otro problema se presenta, tanto en sistemas pequeños como en grandes, cuando éste empieza a crecer y no se quiere desechar lo ya creado. En este caso se modifican los procesos y si no se tiene claro el impacto global, se cae en errores. Es en este punto cuando surge la necesidad de crear representaciones de la solución al problema que nos ayuden a comprenderlo mejor y determinar todas sus características.

Un modelo es una simplificación de la realidad y nos provee de los planos necesarios para la comprensión del problema y del sistema que estamos construyendo. Se usan modelos para la representación de nuestro problema y dependiendo de su tamaño y complejidad, será necesario construir varios modelos para distintas vistas: una general, otra a un nivel más profundo y así hasta llegar a las unidades más básicas del sistema.

Al modelar nuestro problema conseguimos lo siguiente:

- Visualizar cómo queremos que sea el sistema.
- Especificar la estructura y comportamiento del sistema.
- Obtener plantillas guía para la construcción del sistema.
- Documentación de las decisiones que hemos tomado.

El proceso de modelado en un sistema formaliza su construcción, acelera el proceso, facilita su mantenimiento y crecimiento.

1.3.2 PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE (PUDS)

El PUDS (Unified Software Development Process) [1] define un esquema de trabajo adaptable a una amplia gama de tipos de problemas. Es una forma de trabajo que descompone el problema en módulos (componentes) y ataca cada uno de forma específica. Interconecta estos módulos en base a relaciones bien definidas y genera una solución completa.

Éste es un proceso iterativo e incremental que se retroalimenta con cada etapa que se desarrolla y provoca que se regrese a etapas anteriores para definirlas mejor.

El PUDS es guiado por los casos de uso, está basado en la arquitectura y es iterativo e incremental.

Las etapas a seguir son:

Análisis: En la que se identifica la funcionalidad básica del sistema de acuerdo a sus entradas y salidas y lo representa como casos de uso. Donde un actor genera una entrada y otro (o él mismo) recibe una salida de una función particular del sistema. A esto se le llama modelo de análisis.

Se presentan las entidades del sistema en un primer esbozo del diagrama de clases; definiendo la relación entre ellas pero no el detalle de sus métodos y atributos.

Diseño: En esta parte se reafirman los casos de uso en base a lo que se aprendió de la etapa anterior. Esto puede provocar modificaciones en el diagrama de clases, el cual ahora se ve a detalle. Se especifica la interacción entre las clases y se definen los atributos y métodos que conforman cada clase con tanto acercamiento como sea posible. A esto se le llama modelo de diseño.

El último paso del diseño es el modelo de despliegue que define cuál es la distribución física del sistema, cuando se usan diversos nodos en el sistema. Y la forma en que se coloca la funcionalidad del mismo.

Implementación: En la etapa de implementación se busca llevar todos los modelos anteriores a códigos de programación para obtener el software en sí. Esta etapa usaremos el modelo de implementación, en el que se describen los elementos marcados por el modelo de diseño: como las clases. Para ello, indica cómo se hará esta transición de componentes abstractos a archivos de código fuente y programas ejecutables. También especifica la organización que tendrán los componentes implementados en base a las herramientas que provee el entorno que se use para la implementación.

Pruebas: La forma en que se probará el sistema puede haberse planeado desde un inicio, pero no es sino hasta que se tiene el sistema completa o parcialmente implementado que se pueden llevar a cabo pruebas de integración y ejecución; probando la respuesta que emite ante las diferentes entradas que se consideraron en el modelo de casos de uso. En esta etapa se usa el modelo de pruebas, que describe cómo se puede probar cada componente ejecutable y la integración entre ellos. También puede definir la forma de probar elementos específicos del sistema, como la interfaz de usuario. Ya que el modelo de desarrollo es iterativo e incremental, es posible que se use la etapa de pruebas como retroalimentación para etapas anteriores que aún no estén muy bien definidas.

Fase de inicio: Tiene como principal objetivo poner en marcha el sistema ya implementado y probado. Propone la forma en que se distribuirá la solución y, en caso de ser necesario, el esquema de negocio que se seguirá. Involucra la creación de manuales de usuario y técnicos, así como la capacitación del personal que quedará a cargo del sistema.

Para el proyecto actual no se pretende liberar una aplicación, por lo que la fase de inicio no se abordará.

1.3.3 Lenguaje Unificado de Modelado (UML)

UML (por sus siglas Unified Modeling Language) [2] es un lenguaje estándar para construir planos de software que surge a mediados de los 90's como una consolidación de 3 tendencias de modelado que ya se manejaban y cuyos enfoques eran distintos:

- El método de Booch: para diseño y construcción.
- OOSE (desarrollado por Jacobson): para soporte para casos de uso.
- OMT (por Rumbaugh): para análisis y sistemas con grandes cantidades de datos.

El resultado fue un lenguaje unificado de modelado (cómo lo indica su nombre) que cubre las necesidades de proyectos de todo tipo y dimensiones. El lenguaje es orientado a objetos y cubre 4 puntos importantes de los elementos de un sistema de software:

- Visualizar
- Especificar
- Construir
- Documentar

UML posee las características de ser robusto, flexible y con un alto nivel de abstracción, lo cual lo convierte en una buena opción para modelar no sólo sistemas de software sino también para sistemas de vigilancia médica, jurídicos, flujos de trabajo, protocolos, etc.

Está constituido de 3 elementos principales:

- Bloques básicos de construcción (que veremos con más detalle).
- Reglas para combinar los bloques.
- Mecanismos comunes del lenguaje.

1.3.3.1 DIAGRAMAS UML

Un diagrama es una representación gráfica de cualquier combinación de elementos mostrados como un conjunto de nodos y aristas para denotar los objetos y sus relaciones. Se usan varios diagramas para mostrar el sistema completo desde diferentes perspectivas. Así en un conjunto de diagramas, un mismo elemento puede aparecer en varios de estos, representándose a sí mismo.

Para crear diagramas es necesario utilizar los bloques de construcción definidos por el lenguaje. En el apéndice II se presentan estos bloques.

1.3.3.2 TIPOS DE DIAGRAMAS UML

Para descomponer un sistema grande de software, UML define 9 tipos de diagrama:

Diagrama de clases: Para mostrar el conjunto de clases, interfaces, colaboraciones y sus relaciones. Cubre la vista de procesos estática.

Diagrama de objetos: Para modelar los objetos y sus relaciones en instantes de tiempo. Cubre la vista de diseño estática.

Diagrama de casos de uso: Para mostrar los casos de uso y los actores que intervienen. Cubre la vista de casos de uso.

Diagrama de secuencia: Es un diagrama de interacción entre objetos definiendo el orden en que ocurren las comunicaciones entre ellos. Cubre la vista dinámica.

Diagrama de colaboración: Es un diagrama isomorfo al diagrama de secuencia. También cubre la vista dinámica.

Diagrama de estados: *Muestra* una máquina de estados que muestra transiciones, eventos y actividades. Cubre la vista dinámica.

Diagrama de actividades: Es un tipo especial de diagrama de estados para mostrar el flujo de las actividades en el sistema.

Diagrama de componentes: Para mostrar la organización y dependencia entre los componentes del sistema. Cubre la vista de implementación estática.

Diagrama de despliegue: Indica la configuración de los nodos de procesamiento en tiempo de ejecución y los componentes en ellos. Cubre la vista de despliegue estática.

1.4 ORGANIZACIÓN DEL DOCUMENTO

Este trabajo propone el modelo de un API estándar, para proveer a los desarrolladores de aplicaciones en la nube un conjunto de servicios comunes para acceder a la plataforma de su elección y permitir la portabilidad.

Lo restante del documento está organizado como sigue: la sección II presenta el estado del arte del cómputo en la nube y sus APIs así como algunos esfuerzos que se han hecho por estandarizar las nubes; la sección III contiene el planteamiento del problema de portabilidad, incluyendo una demostración de éste y la propuesta para solucionarlo; la sección IV presenta los resultados obtenidos donde se muestra el modelo de análisis, de diseño y de implementación; finalmente se discuten algunas conclusiones y trabajo a futuro en la sección V.

2 ESTADO DEL ARTE

2.1 FUNDAMENTOS DEL CÓMPUTO EN LA NUBE

El cómputo en la nube es una tendencia reciente de desarrollo de aplicaciones en la Web. Da al usuario la posibilidad de liberar sistemas de software sin preocuparse por la infraestructura física rentando equipo en la nube.

Una nube se presenta al usuario como un ambiente computacional, accedido mediante Internet, con recursos de hardware aparentemente infinitos. Físicamente las nubes son grandes "data centers" pertenecientes a compañías específicas que los hacen públicos mediante un esquema de "Pay-as-you-go" (paga lo que uses). La mayoría de ellas usan el concepto de virtualización para dar acceso a los usuarios y ofrecer configuraciones flexibles de hardware virtual.

2.1.1 VIRTUALIZACIÓN

Como en cualquier sistema distribuido, se busca que múltiples recursos sean vistos como uno solo para el usuario. Esta tarea se logra utilizando el concepto de virtualización. El cual consiste en usar herramientas que mapeen procesadores, discos duros, particiones, segmentos de memoria y otros a recursos virtuales aparentemente locales. Este concepto es básico en la creación de nubes.

Los hipervisores (hypervisors) consiguen esta abstracción de los recursos al crear máquinas virtuales (VM), que no son más que emulaciones de una entorno computacional completo sobre el que se monta un sistema operativo funcional. Así, una o varias VMs pueden ejecutarse dentro de un mismo equipo de cómputo físico. Al usar máquinas

virtuales, resulta irrelevante la ubicación real de los recursos que se asignan, tan solo se utilizan como si pertenecieran exclusivamente a ellas. Las VMs dan un ambiente seguro y aislado para la ejecución de las aplicaciones de una nube; protegiendo a otros usuarios y otras VMs. Su hardware virtual pude ser reconfigurado fácilmente por el usuario de la máquina (o por los administradores) y no requiere la intervención de técnicos presenciales. [22]

2.1.2 ORÍGENES DEL CÓMPUTO EN LA NUBE: ESFUERZO IBM GOOGLE

Uno de los primeros pasos que se dieron para llegar al actual cómputo en la nube fue la colaboración entre las compañías Google e IBM en el año 2007 [3]. Para aquel proyecto, ambas compañías conformaron un data center de varios cientos de computadoras, con aproximadamente 1700 procesadores, para ofrecer una herramienta de desarrollo y despliegue de aplicaciones con necesidades de paralelismo. Este data center podría ser accedido mediante Internet usando herramientas de software libre. Estaba orientado a estudiantes de ciencias y la comunidad científica, incorporando varias universidades como: Carnegie Mellon, MIT, Stanford, entre otras.

La colaboración fue oportuna ya que al combinar el potencial para la creación de sistemas que tiene IBM con la experiencia de Google para grandes sistemas Web, se conformó un sistema robusto que permitió realizar pruebas de procesamiento de grandes volúmenes de información con buenos resultados.

El software de plataforma que ofrecían incluía: Sistemas operativos Linux, sistemas Xen, Apache Hadoop y MapReduce. Además de herramientas para el monitoreo de la plataforma.

Este proyecto provocaría un cambio en la forma en que se desarrollarían sistemas pesados Web como las redes sociales y multimedia así como incorporaría nuevos métodos para el desarrollo de cómputo en gran escala.

2.2 CÓMPUTO EN LA NUBE

El Instituto Nacional de Estándares y Tecnología (NIST) [4] define el cómputo en la nube como: "Un modelo para habilitar acceso de red omnipresente, conveniente y sobre demanda a una gama de recursos de cómputo configurables que pueden ser rápidamente provistos y liberados con un mínimo esfuerzo de configuración e interacción por el prestador de servicios".

2.2.1 CARACTERÍSTICAS DEL CÓMPUTO EN LA NUBE

De acuerdo a la definición del NIST, si un servicio de cómputo en la nube es bien provisto, debe cumplir con la serie de características que se listan a continuación:

Servicio sobre demanda: indica que el usuario de la nube debe poder obtener más capacidades de la nube sin necesidad de intervención humana.

Acceso de banda ancha: Las capacidades de la nube deben estar disponibles por métodos de red convencionales y heterogéneos para las necesidades de los diferentes dispositivos actuales con acceso a internet.

Agrupamiento de recursos: los recursos de la nube deben estar disponibles a múltiples usuarios bajo un esquema compartido de renta. Físicamente pueden estar en cualquier ubicación más sin que el usuario tenga control o conocimiento de ello.

Elasticidad rápida: Las capacidades de la nube deben poder crecer y decrecer rápidamente y de forma automática, para solventar las demandas que tengan los usuarios. Usualmente las capacidades parecen ilimitadas.

Servicio medido: Los sistemas en la nube deben controlar y optimizar el uso de los recursos mediante monitoreo. Esto también debe aplicar a que se pueda consultar la utilización de los mismos.

El tipo de nube también es importante ya que define el acceso que tienen los usuarios del mundo a ella. Se tienen nubes privadas que sólo pueden ser utilizadas por la compañía a la que pertenece; nubes comunitarias que son provistas a un grupo de organizaciones y consumidores; nubes públicas, que están disponibles al público en general; finalmente las híbridas, que involucran la mezcla de dos o más de uno de los tipos anteriores.

2.2.2 MODELOS DE SERVICIO

La apertura y elasticidad del cómputo en la nube permite que se entreguen diversos modelos de servicio de acuerdo a las necesidades de los usuarios. Estos conforman también la arquitectura que tiene una nube.

2.2.2.1 SOFTWARE COMO UN SERVICIO (SAAS)

El primer modelo de servicio es el de Software como un Servicio. Se refiere a la capacidad que tiene la nube de que los usuarios utilicen software ya incluido y lo personalicen para hacerlo accesible a sus clientes a través de Internet. El control de estas herramientas no depende del usuario.

2.2.2.2 PLATAFORMA COMO UN SERVICIO (PAAS)

La siguiente capa y modelo es Plataforma como un Servicio y tiene que ver con desarrolladores de software. En esta los usuarios despliegan las aplicaciones que ellos crean o adquieren utilizando algún lenguaje de programación soportado por la nube y sus entornos de desarrollo. Está compuesta por los sistemas operativos, interfaces de programación de aplicaciones (API), documentación y servicios básicos.

2.2.2.3 INFRAESTRUCTURA COMO UN SERVICIO (IAAS)

Infraestructura como un Servicio refiere al uso de los recursos disponibles en la nube: memoria, procesadores, almacenamiento, etc. Estos recursos están disponibles para ser usados por las aplicaciones y herramientas que el usuario implementa.

2.2.2.4 HARDWARE COMO UN SERVICIO (HAAS)

Finalmente, Hardware como un Servicio involucra recursos físicos como servidores, racks, ruteadores, etc. Este modelo de servicio no es mencionado como estándar pero algunos proveedores lo incluyen cuando se le permite al usuario rentar equipo físico dentro de la nube [5].

En la Fig. 1 se presenta el diagrama de la arquitectura de una nube propuesto por el NIST [15]. Este muestra tanto los modelos de despliegue, los de servicio y las características principales de una nube.

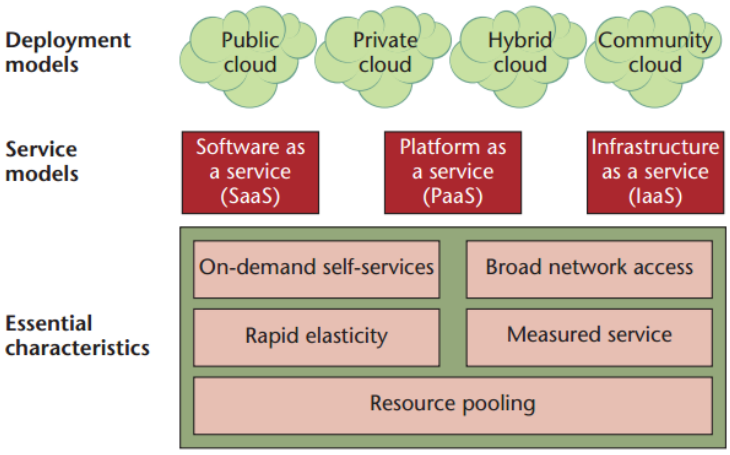


FIG. 1 ARQUITECTURA DE UNA NUBE DE ACUERDO AL NIST

En este trabajo se integra en los modelos de servicio la capa HaaS. Cada proveedor organiza su nube de forma distinta, pero esta arquitectura básica compuesta en capas es válida para las nubes en general, como se ve en la Fig. 2.

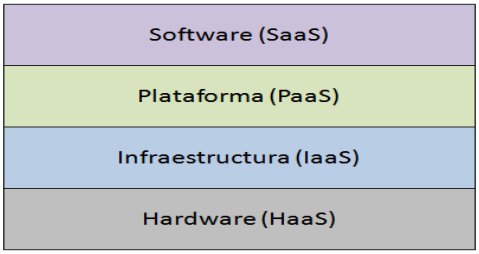


FIG. 2 ARQUITECTURA DE MODELOS DE SERVICO DE UNA NUBE INCLUYENDO CAPA HAAS

2.2.3 APIs COMO HERRAMIENTA DE PLATAFORMA

Las APIs de nubes son bibliotecas de software con un conjunto de procedimientos, funciones y objetos que permiten a los desarrolladores de aplicaciones el acceso y control a la plataforma de la nube y a sus recursos. Estas herramientas son esenciales para obtener acceso completo a las ventajas de la nube. Las APIs son desarrolladas en algún lenguaje de programación particular, siendo los más comunes: Java, Python y C#.

2.2.4 SERVICIOS COMUNES EN LAS NUBES

Cada proveedor de cómputo en la nube ofrece distintos servicios y aborda los problemas desde un enfoque privado. Esto provoca que el usuario de las nubes deba analizar con cuidado qué proveedor satisface sus necesidades: no hay mucho problema si las excede, pero sí si no las cumple su sistema no proveerá los servicios adecuados.

Cuando se habla de los servicios de plataforma se refiere a todos los recursos que brindan los proveedores de las nubes a los desarrolladores de software. Esto para incluir en sus aplicaciones funcionalidad relacionada:

- Base de datos
- Cuentas de usuarios
- Cuentas de correo electrónico
- Unidades de almacenamiento
- Canales de comunicación
- Herramientas para seguridad
- Herramientas para procesar datos
- Cuota de uso de la nube

Estos ocho servicios fueron identificados analizando las características básicas de algunos de los proveedores de cómputo en la nube más usados. A continuación se presenta cada uno de ellos y su funcionamiento básico.

2.2.4.1 SERVICIO DE CUOTA (QUOTA SERVICE)

Cuota se refiere al consumo que ha hecho el desarrollador de los recursos globales de la nube en sus aplicaciones. Ya que el esquema comercial de las nubes permite que sólo se pague lo que se usa, se vuelve importante para el desarrollador saber que no se excede de algún tope que él se establezca y preparar sus sistemas para el caso en que este límite se alcance. Esto le ayudará a controlar el costo que le genera la utilización de sus aplicaciones.

Como un ejemplo podemos hablar de la transferencia de datos. Los proveedores de las nubes definen qué cantidad de bytes transferidos en un intervalo de tiempo (un día, un mes, etc.) corresponden a un costo específico. Amazon cuenta con una tabla bien definida del costo por Gigabyte transferido de acuerdo a la cantidad total del mes. Esto implica que sí tienen las herramientas para monitorear la transferencia de datos y, por lo tanto, es viable consultarla programáticamente.

2.2.4.2 SERVICIO DE ALMACENAMIENTO PERSISTENTE (PERSISTENT STORAGE SERVICE)

Almacenar persistentemente algo se refiere a guardar archivos en una unidad no volátil (como un disco duro). Muchas aplicaciones requieren generar o cargar información de archivos convencionales y manipularlos como lo harían en cualquier otro sistema local.

Algunas nubes soportan el uso de máquinas virtuales que se comportan como una computadora personal o servidor. En éstas el almacenamiento se percibe igual que en cualquier sistema operativo comercial y se puede trabajar con él sin necesidad de integrar una biblioteca de propietario. Desde luego que este esquema tiene la limitante de espacio disponible finito y posiblemente pequeño.

Otro esquema, más atractivo, que ofrecen las nubes es el de almacenamiento elástico. En este caso, se puede almacenar archivos en unidades virtuales que gestiona la nube de forma transparente al usuario. La ubicación real de los archivos y unidades es incierta. La gran ventaja de este esquema es que las nubes prometen una cantidad de espacio disponible infinito, por lo que el programador no se tiene que preocupar por saber si puede o no almacenar más archivos.

Como ejemplo de almacenamiento elástico tenemos las páginas Web que permiten cargar contenido multimedia (video, imágenes, etc.). Cada archivo tendrá un tamaño relativamente alto: si almacena imágenes en alta resolución que en promedio son de más de 1 Mb de tamaño, música de 5 a 10 Mb por archivo y videos de algunas decenas o centenas de Megabytes, y recibe diariamente nuevos archivos, en poco tiempo cualquier disco duro actual será insuficiente. Con el almacenamiento elástico el usuario no se

preocupará por esto pero sí es importante que consulte sus cuotas de almacenamiento para asegurar que no se le dispare el costo y tenga pérdidas monetarias.

2.2.4.3 SERVICIO DE MANEJO DE BASE DE DATOS (DATABASE MANAGEMENT SERVICE)

Las bases de datos son algo muy popular y útil para los sistemas modernos. En especial para aquellos que relacionan alguna clase de información con el usuario que los utiliza. El modelo de base de datos relacional es el más usado.

No todas las nubes ofrecen bases de datos o no lo hacen con un modelo relacional. Aquellas que sí lo hacen necesitarían dar acceso a ellas mediante primitivas estándar que permitiera crear esquemas (bases de datos), tablas, relaciones, etc. Así como las operaciones necesarias para conexión, desconexión, alta, baja y manipulación de registros, entre otras.

Si el manejador de base de datos con el que trabaja la nube no es relacional, debe incluir una capa por encima de sus primitivas que den la apariencia de ser un sistema relacional (como es el caso de Google y su manejado Bigtable), tratando de no perder eficiencia y elasticidad.

2.2.4.4 SERVICIO DE SEGURIDAD (SECURITY SERVICE)

La seguridad de la información que se maneja en Internet así como su confidencialidad es un elemento clave. Protegerla de intrusos que intenten leerla o manipularla es un problema serio de criptografía y lo mínimo a cubrir son los siguientes puntos:

Autenticación: verificar la supuesta identidad de un usuario o sistema.

Control de Acceso: proteger los recursos del sistema contra accesos no autorizados.

Confidencialidad: proteger los datos contra revelación no autorizada.

Integridad: proteger los datos contra modificaciones no autorizadas, inserciones o borrado.

No-repudio: proveer protección sobre que el emisor de un mensaje niegue haberlo enviado y contra que el receptor niegue haberlo recibido.

El programador buscará cifrar su información confidencial mediante algoritmos de cifrado simétricos o asimétricos, como: AES, ECDSA, RSA. Este es un tema muy importante en cómputo en la nube pues se ha hablado mucho sobre la inseguridad de los datos que ésta presenta. También asegura canales de comunicación para establecer una comunicación

privada entre sus mismas aplicaciones o con las de otros. En este caso se puede usar conexiones seguras con SSL.

Protege sus datos de modificaciones no autorizadas usando algoritmos de códigos HASH y MAC. Como ejemplo se tiene SHA2. Comprueba el origen de los datos para evitar falsificaciones mediante firmas digitales y hace uso de certificados.

2.2.4.5 Servicio de comunicación (communication service)

La comunicación entre una aplicación y otra es una tarea de lo más común. Ya sea que ambas estén dentro de la misma nube o una distinta; que las dos le pertenezcan al mismo programador o no, la comunicación se debe de dar de forma constante y consistente.

2.2.4.6 Servicio de manejo de usuarios (user management service)

El cómputo en la nube se usa con mucha frecuencia en el desarrollo de aplicaciones Web para usuario final. Conforme estas aplicaciones se vuelven más robustas, se busca la forma de otorgar personalización de las características del sitio, si es que no era el objetivo principal. Eventualmente caen en la necesidad otorgar credenciales al usuario con las que se autentica para ingresar a partes del sistema que son exclusivamente de él. Lo más básico para estas credenciales es tener un usuario registrado.

Un usuario contará con la información más relevante para el sitio, de forma que pueda ser usada para personalizar el contenido. La vista frontal de muchos sistemas de software en la nube es presentada al usuario como una aplicación Web y las cuentas de usuario son algo básico en este paradigma.

2.2.4.7 Servicio de acceso a E-mail (e-mail access service)

Al tener cuentas de usuario, un sistema de e-mail resulta útil para comunicarse con ellos.

2.2.4.8 Servicio de procesamiento de datos (data processing service)

Necesario para tomar ventaja de los recursos ilimitados de una nube al procesar información compleja o grandes cantidades de datos.

2.3 Estandarización en las nubes

Tal como la tendencia de cómputo en las nubes crece, algunos esfuerzos de estandarización se han realizado por diferentes entidades, tal como: Definición de Cómputo en la Nube [4] y Sinopsis y Recomendaciones para el Cómputo en la nube [15], por el NIST; Perfiles de nubes [23] e Intercloud [24], por el IEEE y Cómputo en la nube y estandarización: Reportes técnicos [25], por el ITU-T. Todos ellos abordan el tema de la

estandarización proponiendo que se solucionen los problemas de interoperabilidad y arquitectura. En cuestión de portabilidad aún no se tiene una propuesta estándar.

Otro de ellos es el Formato de Virtualización Abierto (OVF) [6], que involucra la homogenización de la estructura de máquinas virtuales, permitiendo transportarlas entre nubes sin preocuparse del hipervisor del proveedor.

Otro estándar es propuesto por la Incubadora de Estándares de Nubes Abiertas [7]. En la publicación Nubes Interoperables ellos atacan el problema de la interoperabilidad entre nubes. Proponen que los proveedores de nubes deben incorporar servicios estandarizados para la intercomunicación. El *Open Grid Forum* [8] también sugiere estándares para la interoperabilidad entre nubes. Desafortunadamente ninguna de las propuestas es actualmente un estándar internacional.

El *Open Cloud Manifesto* [9] promueve que las implementaciones de las nubes deben usar lineamientos, pero no propone una solución concreta.

2.3.1 JCLOUDS

jClouds [10] es una implementación en Java de un paquete estándar para unificar múltiples proveedores de nubes. Soporta las nubes más populares y da a los desarrolladores la posibilidad de mantener sus aplicaciones libres de lock-in. Puede que éste sea lo más cercano a lo que se propone este documento, pero su estrategia es diferente. En la Fig. 3 se muestra la ubicación de jClouds en la arquitectura de una nube.

JClouds es un API no estándar colocada entre la aplicación y el API de la nube. Sus desarrolladores se dan a la tarea de incorporar soporte para cada nube popular. Debido a esto, jClouds tiene un problema potencial de crecimiento; entre más nubes surjan, el API se volverá más y más grande hasta volverse demasiado pesada o tendrá que ignorar algunas nubes para mantenerse ligera. Y como no hay compromiso de las nubes a jClouds, un proveedor podría cambiar su API y afectar el resultado de esta biblioteca. jClouds es la única manera de crear aplicaciones portables en la nube en este momento. Es una buena estrategia pero puede no ser suficiente a futuro.

A diferencia de jClouds, el modelo del API que se está desarrollando no pretende ser una biblioteca adicional a la del proveedor, sino que la de él sea implementada basándose en un estándar para que sea compatible con las de otras nubes.

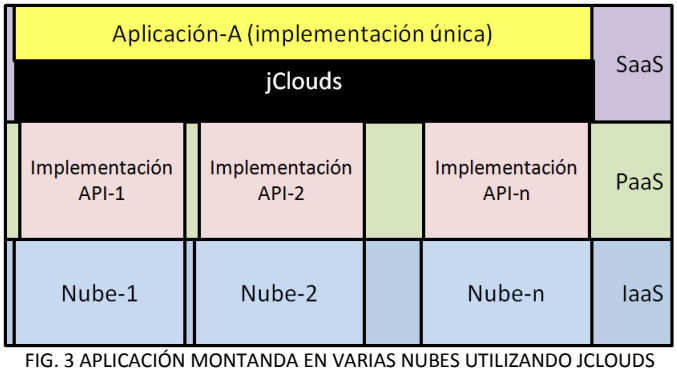


FIG. 3 APLICACIÓN MONTANDA EN VARIAS NUBES UTILIZANDO JCLOUDS

2.3.2 OTROS ESTÁNDARES

Poniendo a un lado las nubes, las APIs estándar son una solución poderosa y elegante para unificar funcionalidad de tecnologías con múltiples implementaciones.

Tomemos como ejemplo a Java y las bases de datos. Existe una API estándar llamada "*Java Data Base Connectivity*" (comúnmente conocida como JDBC) [11]. Este paquete surge como una implementación del ya existente del API estándar *Open Database Conectivity* (ODBC) que brinda la posibilidad de conectar aplicaciones, en diversos lenguajes de programación, con bases de datos relacionales. JDBC permite a los desarrolladores en Java interactuar con distintos proveedores de bases de datos (Oracle, MS SQL, MySQL, etc.). No maneja la comunicación real entre la aplicación y la base de datos, sólo contiene clases básicas e interfaces para definir el comportamiento de objetos estándar sin implementarlo.

El desarrollador incluye un driver con la implementación adecuada de JDBC para la base de datos deseada. Esto le da la libertad para migrar de un manejador de base de datos a otro cambiando el driver y ajustando ligeramente el código. JDBC ha sido la solución para bases de datos con Java por muchos años y es un buen ejemplo de lo que un API estándar bien planeada es.

2.4 NUBES COMERCIALES

Para este trabajo, se usan dos proveedores de cómputo en las nubes como base. Éstos fueron escogidos por su popularidad en el mercado y para conseguir que dos esquemas muy distintos de trabajo trabajen de una misma forma.

Las dos nubes elegidas fueron *Amazon Services (AWS)* [12] y *Google AppEngine* [13]. Otras nubes, como *Microsoft Azure* e *IBM Blue Cloud* trabajan en un esquema similar a Amazon, por lo que se usa como un representante de este tipo de cómputo en la nube basado en virtualización de sistemas operativos.

2.4.1 AMAZON AWS

La compañía Amazon (famosa por su venta de libros por Internet) ofrece el servicio de cómputo en la nube. *Amazon Web Services (AWS)* es el nombre de su conjunto de servicios.

Dentro de ellos se encuentra *Amazon Elastic Compute Cloud (Amazon EC2)* que es un entorno virtual de cómputo muy estable y fácil de manejar.

Ellos describen su servicio de nube de la siguiente forma "*Amazon Elastic Compute Cloud (Amazon EC2) [12] es un servicio web que proporciona capacidad informática con tamaño modificable en la nube. Está diseñado para facilitar a los desarrolladores recursos informáticos escalables y basados en web*".

Su nube permite el uso de la infraestructura pública de Amazon hacia usuarios que requieran sistemas de cómputo flexibles. Tiene una excelente capacidad para configurar, asignar y liberar recursos de forma dinámica. El pago del servicio se maneja mediante esquemas de paga lo que uses, con lo que resulta muy económica su utilización, si se sabe administrar adecuadamente.

Con este servicio se ofrecen plataformas virtuales con sistemas operativos comerciales, lo que le da al usuario la percepción de poseer un servidor físico que puede controlar desde cualquier parte del mundo. Estas plataformas virtuales se conocen como Amazon Micro Instances (AMI) y se acceden mediante escritorios virtuales o conexiones SSH.

El trabajar máquinas con virtuales facilita mucho el trabajo, ya que el desarrollador de aplicaciones se enfrenta a un entorno familiar y amigable sobre el que instala la paquetería que requiera así como sus propias aplicaciones. Con esto, las aplicaciones que el usuario implementa en AWS pueden ser tanto Web como de escritorio.

2.4.1.1 CARACTERÍSTICAS DE AMAZON AWS

Amazon ofrece las siguientes características en su nube, con las que se permite crear aplicaciones flexibles con necesidades de ajustes rápidos de sus recursos.

Amazon elastic block store: Es una forma de almacenamiento persistente que no depende de una instancia de EC2, con lo que se puede utilizar en una o varias instancias.

Varias ubicaciones: Las ubicaciones física, regional de las instancias causan impacto en el costo que tiene al igual que la disponibilidad que tendrán. Amazon cuenta con zonas

aisladas de fallos, con tolerancia a baja latencia en la comunicación y también para colocar instancias cerca de alguna región sobre la que se desea trabajar.

Direcciones Elastic IP: Son dirección IP estáticas provistas por Amazon que se pueden asignar de forma dinámica a un instancia u otra, con lo que se consigue transparencia al migrar un servicio de una instancia a otra.

Amazon Virtual Private Cloud: Es una red privada virtual que permite interconectar de forma estable y segura la infraestructura de una compañía con la de Amazon a fin de trabajar en grupo.

Amazon CloudWatch: Permite la supervisión de los recursos y aplicaciones que se montan en la nube. Esto permite medir el consumo de red, CPU, almacenamiento, etc.

Auto Scaling: Es la capacidad para incrementar o reducir el tamaño de una instancia dinámicamente de acuerdo a las necesidades de las aplicaciones que maneja. Permite que se generen nuevas instancias sobre demanda y se liberen cuando ya no son necesarias.

Elastic Load Balancing: Permite que el tráfico entrante se distribuya entre un conjunto de instancias para ser atendido sin problemas de sobrecarga. Identifica aquellas instancias que tienen problemas o demasiado trabajo y balancea automáticamente a otras en mejor estado.

Clústeres de Computación de alto rendimiento (HPC): Permite la implementación de procesos que requiera fuerte paralelismo y eficiencia. Provee nodos estrechamente conectados para una latencia baja en la comunicación.

VM Import: Permite importar imágenes de equipos virtuales que el usuario tiene localmente a instancias EC2, aprovechando el esfuerzo que se haya invertido en configurar estos sistemas previos.

2.4.2 GOOGLE APPENGINE

La compañía Google (del popular buscador de Internet) tiene su servicio de cómputo en la nube y lo maneja de una forma distinta a otros. El nombre de su nube es Google AppEngine.

En el sitio oficial de AppEngine [13], se describe el servicio de la siguiente forma “Google AppEngine permite crear y alojar aplicaciones web en los mismos sistemas con los que funcionan las aplicaciones de Google. Google AppEngine ofrece procesos de desarrollo y de implementación rápidos, y una administración sencilla, sin necesidad de preocuparse por el hardware, las revisiones o las copias de seguridad y una ampliación sin esfuerzos”.

La nube de Google no trabaja con sistemas operativos virtualizados, por lo que no se tiene un entorno al que se pueda acceder como a un equipo remoto. En lugar de eso, Google ofrece un contenedor de aplicaciones Web que se le pueden cargar mediante herramientas provistas por el mismo proveedor.

El entorno de Google permite tener aplicaciones Web que nunca se queden sin recursos disponibles, con lo que la flexibilidad y escalamiento se resuelven de forma automática. Esta característica hace que la nube de Google sea altamente confiable para sistemas grandes sin tener que preocuparse mucho por los picos de uso. Al igual que otras nubes, Google ofrece el sistema de paga lo que uses, con lo que las cuotas se pueden mantener muy bajas y variar de acuerdo al tamaño del sistema que se implementa.

2.4.2.1 Características de Google AppEngine

Servidor web dinámico: Ofrece un contenedor Web con las características convencionales de otros en el mercado como Apache Tomcat. Esto permite migrar aplicaciones previamente usadas en otros contenedores ajustándolas a la infraestructura de Google.

Almacenamiento permanente con funciones de consulta, clasificación y transacciones: Permite almacenar tanto archivos como datos, consiguiendo tener tanto un sistema de archivos (no convencional) como un sistema de base de datos (no relacional)

Escalado automático y distribución de carga: permite que las aplicaciones tomen y liberen recursos dinámicamente. También se pueden lanzar varias instancias de una misma aplicación para soportar el alta demanda en su uso.

API para autenticar usuarios y enviar correo electrónico a través de Google Accounts: permite que se utilicen las cuentas de usuario y correo electrónico de Google (gmail) para autenticar a los usuarios del sistema sin tener que preocuparse por su gestión.

Completo entorno de desarrollo local que simula Google AppEngine un equipo local: Permite realizar pruebas controladas de los sistemas desarrollados en una computadora local, agilizando y facilitando el proceso de liberación. Consta de un contenedor Web similar al real que se utiliza como plataforma de despliegue.

Colas de tareas que realizan trabajos fuera del ámbito de una solicitud web: Es una forma de crear tareas que se ejecutan sin la necesidad de un front-end Web que las haya invocado. Se pueden programar para ejecutarse en momentos específicos o con intervalos.

3 PLANTEAMIENTO DEL PROBLEMA

3.1 EL PROBLEMA DE PORTABILIDAD

Muchas nubes ofrecen servicios similares como almacenamiento persistente y manejo de bases de datos entre otros. Un desarrollador accede a los servicios del API dada por el proveedor de la nube. Pero cada nube es distinta y también lo son sus APIs. Este es un problema grave para el desarrollo de aplicaciones en la nube.

Cuando una aplicación es creada sobre una nube, usando el API provista, cae en un concepto llamado *lock-in* [14] donde la aplicación está atada a la infraestructura donde fue creada. Esto es problemático por los desarrolladores ya que tendrá problemas al tratar de transportar su aplicación de una nube a otra debido a la incompatibilidad entre APIs.

Algunos esfuerzos se han hecho para estandarizar las nubes, desgraciadamente ninguna de las propuestas se ha aceptado como un estándar internacional y, por otro lado, los proveedores no quieren perder sus clientes.

El NIST [15] menciona esta problemática tanto en los modelos SaaS como PaaS. Más adelante se muestra con un experimento el problema en cuestión.

Este trabajo propone una capa abstracta, llamada API estándar, que provee a los desarrolladores de aplicaciones en la nube un conjunto de servicios comunes para acceder al API pública de la nube de su elección. El API estándar es desarrollada usando Proceso Unificado de Desarrollo de Software y su Lenguaje Unificado de Modelado.

Entre nubes públicas la diferencia entre las APIs puede ser enorme. Tomemos el AppEngine de Google y EC2 de Amazon. La primera es orientada a un contenedor Web,

mientras que la segunda es orientada a esquema de escritorio. Esto hace muy complicado el transportar una aplicación de una de ellas a la otra.

Una manera de circular este problema podría ser instalando un contenedor Web en la nube EC2 para tener un acercamiento similar en ambas. Pero eso no es suficiente; la diferencia entre el API de AppEngine y otra persiste y la dependencia de la aplicación a ella obliga al desarrollador a ajustar el código original de la aplicación a la nueva nube. A mayor dependencia, mayor dificultad para portar la aplicación.

3.2 PROPUESTA

Una solución posible a los problemas de portabilidad podría ser la estandarización de un conjunto de servicios comunes ofrecidos por las nubes. Estos servicios homogenizan la interacción entre APIs y define un API abstracta estándar.

Así, esta API necesita ser implementada por cada proveedor de nube, dando acceso a su infraestructura particular de una forma general. El desarrollador de aplicaciones usa el API estándar sin saber cómo funciona ni preocuparse por qué nube hizo la implementación. Cuando quiera migrar de una nube a otra, tan sólo cambia la implementación del API y tal vez haga un ligero ajuste en su código para tenerlo trabajado nuevamente, como se ve en la Fig. 4.

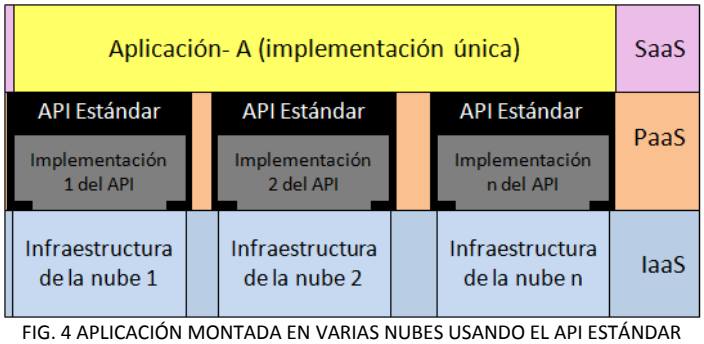


FIG. 4 APLICACIÓN MONTADA EN VARIAS NUBES USANDO EL API ESTÁNDAR

Algunas nubes ofrecen servicios únicos que no pueden ser estandarizados. Esta es una dependencia potencial entre la nube y la aplicación. Puede que siempre sea un problema pero el usar una API estándar en vez de las actualmente provistas lo reduce al mínimo.

3.3 EJEMPLO DEL PROBLEMA (JUSTIFICACIÓN)

A manera de justificación, se realizó un experimento comparativo entre dos proveedores de nube para comprobar que la realización de una misma tarea no se puede lograr de la misma forma.

Se tomaron las nubes de *Amazon EC2* y *Google AppEngine*.

Las metas a cumplir con este ejemplo son las siguientes:

1. Crear un archivo de texto con el nombre *series1000.txt* que contenga texto con los números del 0 al 999.
2. Almacenar el archivo de forma persistente en las dos nubes dentro de un subdirectorio llamado *example*.
3. Volver a cargar el archivo y leer su contenido.

3.3.1 IMPLEMENTACIÓN DEL EJEMPLO

Cada una de las nubes en cuestión brinda un kit de desarrollo para acceder a sus recursos. El manejo del almacenamiento es similar pero su tratamiento no lo es. Debido a algunos conflictos con la versión 1.7 de *Java* se optó por utilizar el *JDK 6* (*Java 1.6*).

Se trabaja con dos tipos de almacenamiento. El primero es el local en el *AMI* de *Amazon* que trabaja de la misma forma y con las mismas limitaciones de una computadora cualquiera. La segunda es el almacenamiento elástico que es una de las virtudes que posee el cómputo en las nubes. Consiste en almacenar archivos en unidades conocidas como *buckets* (cubetas) sin saber dónde están físicamente los archivos ni si están en un mismo lugar, además de no preocuparse por el espacio disponible.

3.3.2 IMPLEMENTACIÓN EN AMAZON AWS

Amazon AWS con EC2 ofrece la posibilidad de usar máquinas virtuales que simulan el comportamiento de una computadora completa con algún sistema operativo conocido (versiones de *Linux* y *Windows*) a las que llaman *AMIs* (*Amazon Micro Instance*). Para esta demostración se usa una Micro instancia *Linux* de *Amazon EC2*. Esta posee 8GB de almacenamiento convencional. El ejemplo se realizó con dos modalidades de almacenamiento admitidas.

3.3.2.1 IMPLEMENTACIÓN CON ALMACENAMIENTO CONVENCIONAL

Gracias a la virtualización, se pudo acceder al sistema de archivos de *Linux* y se realizó la tarea del ejemplo usando las herramientas básicas de *Java*, incluidas en el paquete *java.io.* Se creó un archivo local dentro de su correspondiente directorio y se escribió a él utilizando un *FileOutputStream* convencional.

El resultado fue la correcta ejecución del código implementado, se generó correctamente el archivo mediante un *Servlet* y se volvió a cargar sin problemas mostrando su contenido en otro *Servlet* más.

3.3.2.2 IMPLEMENTACIÓN CON ALMACENAMIENTO ELÁSTICO S3

Amazon también ofrece el servicio de almacenamiento elástico al que llama *S3* (Simple Storage Service). En *S3* es posible usar en el nombre del archivo una ruta de tipo jerárquica con la que se simula el uso de directorios. Para esta parte del ejemplo sí fue necesario utilizar el API provista por *Amazon*: *AWS SDK 1.3.7* [16].

Para generar el archivo fue necesario crearlo primero de forma local usando *java.io* para subirlo al *bucket* posteriormente mediante una solicitud de carga de archivos *PutObjectRequest.* El cargado del archivo también fue correcto y más directo, ya que *S3* permite abrir una conexión directa (*InputStream*) para lectura del contenido de los archivos en el bucket.

Para acceder al bucket de la nube es necesario autenticarse mediante credenciales ya que se pueden usar los servicios de *Amazon* desde fuera de la infraestructura.

3.3.3 IMPLEMENTACIÓN EN GOOGLE APPENGINE

La nube de *Google* se maneja mediante un contenedor Web desarrollado por el mismo proveedor. No existe el concepto de máquinas virtuales. Tampoco cuenta con un servicio de almacenamiento convencional. El servicio de almacenamiento es elástico, al igual que en *S3*, sólo que en esta ocasión el archivo se crea mediante una petición al servicio y escribe directamente en el archivo abriendo un canal (*FileWriteChannel)* el cual se tiene que cerrar dos veces: una para terminar el flujo y una segunda para que el archivo no pueda ser modificado. Para desarrollar el ejemplo fue necesario utilizar el API de Google: *AppEngine SDK 1.6.4* [17].

Mediante un Servlet, se creó el archivo con el contenido necesario. Un Servlet más se implementó para hacer el cargado del archivo, esto fue exitoso.

3.3.4 COMPARATIVA DE CÓDIGO FUENTE

En esta comparativa se muestra el código fuente de las clases implementadas para el ejemplo en cada una de las plataformas.

Se resalta con recuadros las zonas de código que están en conflicto y previenen la portabilidad. Las líneas continuas marcan el código para crear el archivo, mientras que la línea punteada, las líneas de código que se usaron para escribir el contenido del archivo. En la Fig. 5 aparece el código fuente de la clase para la nube de *Amazon* a la izquierda con almacenamiento local. Al centro se tiene el código de la clase análoga en la nube de *Google*. Al lado derecho de la misma figura, se tiene otra la clase para el servicio *S3*.

FIG. 5 COMPARATIVA DE CÓDIGO AL CREAR UN ARCHIVO DE TEXTO CON EC2 Y APPENGINE.

Finalmente se observa el encabezado de las clases que se usaron para la carga del archivo en el tercer punto del experimento. Se ve claramente la dependencia del código a las nubes particulares.

La propuesta que se hace pretende liberar de esa dependencia a las clases usadas en las aplicaciones en la nube, permitiendo que puedan ser portadas fácilmente de una plataforma a otra. En la Fig. 6 se muestran los encabezados de la clase para EC2 (arriba), AppEngine (en medio) y un prototipo de lo que sería el encabezado de una clase independiente del proveedor de nube (abajo).

FIG. 6 IMPORTS DE LAS CLASES PARA CARGAR EL CONTENIDO DEL ARCHIVO

3.3.5 RESULTADOS DEL EJEMPLO

En la Tabla 1 se presentan datos técnicos de la forma en que se resuelve cada caso del ejemplo. De aquí podemos concluir lo siguiente:

Aunque sí es posible realizar todas las operaciones esperadas en ambas nubes, la diferencia en el código demuestra que los cambios en una migración serían significativos y costosos. La situación más impactante es la importación de clases y paquetes que, claramente, son de APIs de propietario y no serían compatibles al migrarse.

Queda claro que no hay compatibilidad entre estos dos proveedores de nube y no es sencillo portar aplicaciones entre ellas.

TABLA 1 COMPARATIVA TÉCNICA PARA LAS TRES IMPLEMENTACIONES.

Caso	Amazon local	AppEngine	Amazon S3
Crear la instancia del archivo	Se utiliza la clase *File* de java.io que es bien conocida en el desarrollo en Java.	Se obtiene una instancia de un *AppEngineFile* solicitando la creación en el bucket al servicio.	Se debe crear el archivo localmente para colocarlo en el bucket después.
Escribir el contenido al archivo	Se escribe el texto directamente abriendo un *FileOutputStream* al archivo.	Se debe manejar como un tipo *FileWriteChannel* que se abre a partir de una instancia ya creada en el bucket y luego se le abre un flujo.	Se llena un archivo temporal de la misma forma que se hizo con el local y posteriormente se coloca en el bucket.
Leer el contenido del archivo	Se abre un flujo con la clase FileInputStream y se puede leer directamente del archivo.	Se debe solicitar la apertura de un *FileReadChannel* al archivo en el bucket y luego se le abre un flujo de lectura normal.	Se solicita la apertura directa a un archivo en el bucket lo que devuelve un *InputStream* ya listo.

4 MODELOS DEL API ESTÁNDAR (RESULTADOS OBTENIDOS)

4.1 MODELADO DEL API ESTÁNDAR

La finalidad de este trabajo es proponer la estructura básica de un API estándar para cómputo en la nube siguiendo la metodología del Proceso Unificado de Desarrollo de Software y usando el Lenguaje Unificado de Modelado. La flexibilidad de este proceso permite que sea apto para el presente proyecto.

Ya habiendo presentado el marco teórico sobre el cómputo en las nubes y el problema de portabilidad existente, se modela el API propuesta. Se identificará la interacción con sistema en el modelo de análisis, así como las clases que lo conforman. Posteriormente se concretará la interacción entre las clases y los elementos que contienen en el modelo de diseño y finalmente se llevará a código fuente una versión del API en el modelo de implementación.

4.2 MODELO DE ANÁLISIS DEL API ESTÁNDAR

La primera etapa que se aborda es el análisis; en ella se identifica a los usuarios y las interacciones que tienes los usuarios con el sistema. A estas interacciones las llamaremos casos de uso. Empezamos por definir a los usuarios involucrados en el desarrollo de aplicaciones para cómputo en la nube.

En este modelo, se presenta el diagrama de casos de uso que muestra la interacción entre los usuarios y el sistema. También se incluye una primera versión del diagrama de clases, que contiene únicamente la organización y definición de éstas sin entrar en detalle.

4.2.1 USUARIOS DEL SISTEMA

Los usuarios del sistema son aquellos sujetos que tienen interacción con el sistema. Tanto pueden ser personas reales como otros sistemas que hagan uso del que se está diseñando. A continuación se presentan los usuarios que se identificaron del API estándar.

4.2.1.1 DESARROLLADOR DE APLICACIONES

El primer usuario que se puede incluir es el desarrollador de aplicaciones. Él es quien accede de forma remota a la plataforma de la nube que eligió para hacer uso de los recursos que ofrece la infraestructura. Escoge cuáles de los recursos necesita para su aplicación, puede ser: cuentas de correo, seguridad en sus datos, transferencia de información, etc. Programa aplicaciones usando las herramientas brindadas por el proveedor de la nube.

Muchas de las aplicaciones que crea van orientadas a usuarios finales con algún fin comercial en dos presentaciones muy comunes: las *Web* y los *Web services*. Aunque no son las únicas formas de sistemas ni objetivos: también puede programar para fines científicos que exploten el "ilimitado" poder de cómputo de la nube; también hace complementos para aplicaciones de escritorio que los usen para compartir datos.

Actualmente un desarrollador programa aplicaciones usando las herramientas brindadas por el proveedor de la nube. Esto implica que atará su sistema a la paquetería de propietario que necesita. Podría tratar de mantenerse usando sólo paquetería estándar, pero perdería la capacidad para explotar la infraestructura sobre la que está trabajando.

El desarrollador de aplicaciones se enfrenta a la tarea de ajustar el código cuando se migra una aplicación de una nube a otra. Él es quien se ve más afectado por la incompatibilidad entre plataformas y quien más ayuda recibiría de un API estándar.

4.2.1.2 API PÚBLICA

En la capa de plataforma de la nube, se encuentran los elementos que da el proveedor para explotarla. El API pública es la herramienta que brinda para controlar sus recursos. Es una biblioteca de propietario que contiene dos tipos de elementos: clases y recursos.

Las clases que contiene están programadas para controlar la infraestructura. Cómo están hechas no es de incumbencia para el desarrollador. La organización de éstas y el manejo de versiones dependen completamente del propietario. El desarrollador hará invocaciones a estas clases confiando en que obtendrá el resultado prometido por la documentación.

Los recursos son archivos adicionales que contiene el API pública y que le sirven al desarrollador como complementos en sus aplicaciones al momento de utilizar la plataforma. Algunos de los recursos puede ser: multimedia, certificados, textos libres, etc.

Las API públicas actuales de las nubes no siguen ningún lineamiento en cuanto a la estructura que tienen sus clases. No hay un acuerdo en la forma en que se llamará la clase que resuelve las peticiones para bases de datos o para el manejo de usuarios, por ejemplo, ni los métodos que ésta incluirá. Es en este rubro donde recae todo el problema de la portabilidad. Al no tener estándares, las aplicaciones no embonan por sí solas al cambiarlas de nube y sustituir el API pública de una por otra.

4.2.2 CASOS DE USO (INTERACCIÓN CON EL SISTEMA)

Ya definimos a los usuarios que están involucrados en el sistema, ahora procedemos a identificar cómo interactúan estos dos. Estas interacciones serán las necesidades que tiene el desarrollador de aplicaciones y que trata de solucionar mediante peticiones para el API pública a través del API estándar.

Se observa que, de forma muy amplia, la funcionalidad de la plataforma de una nube está organizada en servicios. Estos son agrupaciones de funciones que permiten al desarrollador controlar un recurso muy particular. Para cada uno de los servicios definiremos un caso de uso.

Servicio de Cuota (quota service): Este servicio debe proveer mecanismos para consultar en cualquier momento la cuota de uso de cada uno de los otros recursos de la nube. Debe poderse definir cuál es el tope esperado en relación a costo o las unidades que maneje el recurso. Quizá incluso instalar alguna clase de disparador que notifique a la aplicación cuando se ha llegado a una marca.

Servicio de almacenamiento persistente (persistent storage service): El servicio de almacenamiento persistente debe contar con las funciones necesarias para manipular unidades de almacenamiento elástico. Esto implica operaciones como crear y destruir archivos, así como trabajar sobre su contenido. También se esperaría que sea posible organizarlos jerárquicamente en directorios (carpetas) para facilitar su manejo así como listar el contenido de estos directorios. Más adelante se hablará de este servicio a detalle.

Servicio de manejo de base de datos (database management service): Este servicio debe permitir el manejo de bases de datos relacionales sin importar el esquema que la nube maneje. Esto implica la inicialización, conexión, consulta y modificación de las bases de datos y su estructura.

Servicio de seguridad (security service): Para todas las operaciones de seguridad servicio debe proveer los mecanismos necesarios. Puede que los implemente él en su propia biblioteca o que incluya la de un tercer participante. Lo importante es que el programador de aplicaciones las utilice sin preocuparse por su origen y siguiendo un modelo estándar.

Servicio de comunicación (communication service): En este servicio se incluye la apertura de canales de comunicación como los sockets que habiliten la transferencia de datos de forma monitoreada. También podría incluir un conjunto de datos estándar para facilitar el entendimiento entre aplicaciones de distintas fuentes: XML es un tipo de dato actualmente estándar en la nube.

Servicio de manejo de usuarios (user management service): El servicio implementa procedimientos de registro y login, liberando al desarrollador de aplicaciones de estas tareas. El desarrollador accede a un conjunto limitado de información de usuarios previamente creados y confía en la seguridad de la nube.

Servicio de acceso a E-mail (e-mail access service): En este caso, la nube aloja servidores de e-mail para ser usados por la aplicación y por el mismo usuario y provee acceso a sus características desde este servicio.

Servicio de procesamiento de datos (data processing service): El servicio incluye herramientas distribuidas cooperativas, como Map-Reduce, para ser usadas por el desarrollador.

El acceder y controlar un de estos recursos será la interacción de la que se habla en el sistema y se maneja como un servicio para visualizarla en su forma global. Cada servicio tendrá su propio conjunto de clases, protocolos y recursos, por lo que se necesita un modelo de cada uno de ellos. El presente trabajo sólo se adentra en el servicio relacionado con el almacenamiento, el resto quedan únicamente indicados.

Ahora se describe cada uno de los servicios necesarios para manejar los recursos identificados.

4.2.3 DIAGRAMAS DE CASOS DE USO

El diagrama de casos de uso involucra a los dos actores previamente identificados:

- El *desarrollador de aplicaciones* es quien solicita servicios a la nube. Construye aplicaciones usando las herramientas que le brinda el proveedor.
- El *API pública* es el conjunto de servicios presentado al desarrollador como una caja negra. Implementa la interacción real con el proveedor de nube.

La Fig. 7 muestra el diagrama de casos de uso del API Estándar.

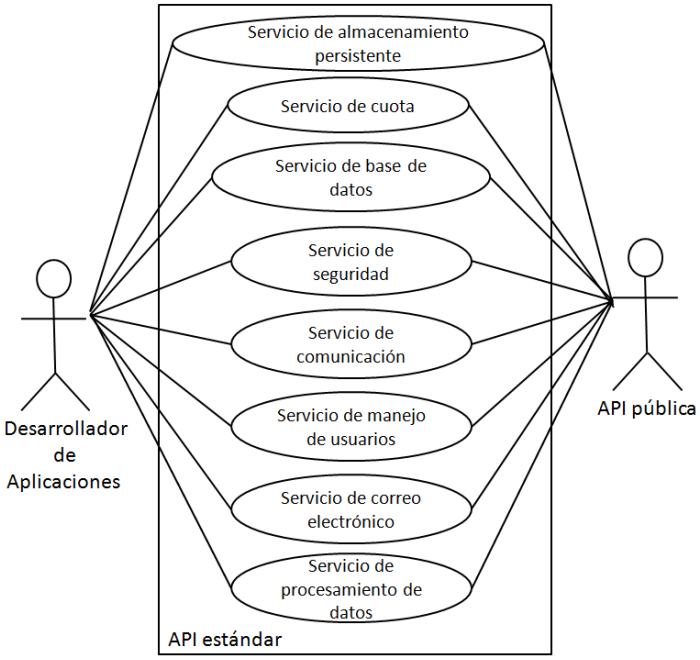


FIG. 7 DIAGRAMA DE CASOS DE USO DEL API ESTÁNDAR

4.2.4 DIAGRAMAS DE CLASES

En el modelo de análisis, el diagrama de clases modela la implementación de servicios sin detalles pero presenta su organización. La Fig. 8 muestra este diagrama.

El API Estándar es presentada como una clase pero sólo para agrupar las clases e interfaces del sistema.

El paquete *PublicAPI* representa la implementación real del API que cada nube pública provee. Su contenido no es de importancia para el modelo puesto que cada proveedor tiene permitido implementar a su gusto.

La clase *CloudServiceException* es la clase padre de todos los errores arrojados por los servicios del API. Pretende ilustrar cómo el API incluye entidades utilitarias para ser más robusta.

El resto de las interfaces coincide con cada uno de los ocho servicios mostrados en el diagrama de casos de uso.

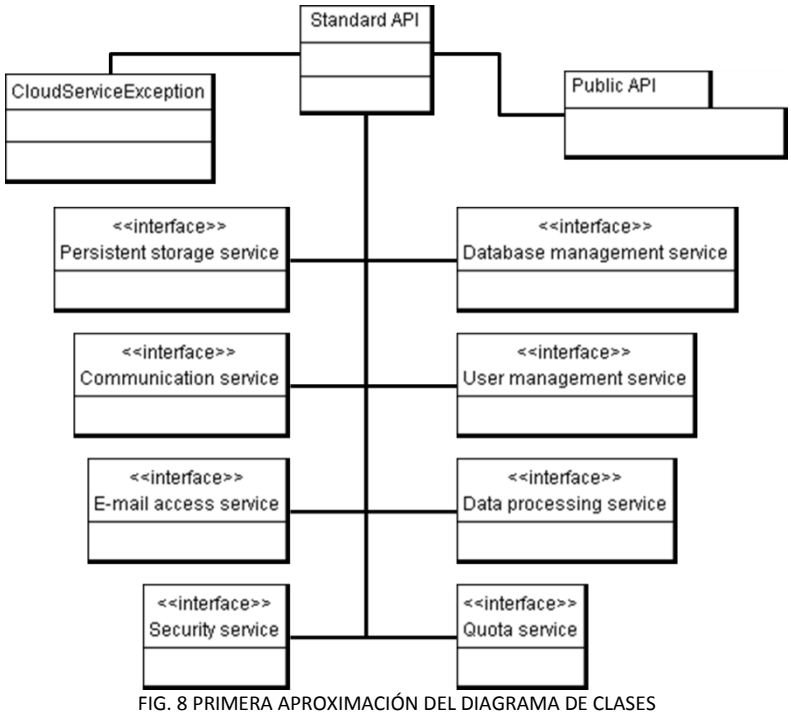


FIG. 8 PRIMERA APROXIMACIÓN DEL DIAGRAMA DE CLASES

4.3 MODELO DE DISEÑO DEL API ESTÁNDAR

Este modelo está compuesto de diversos diagramas, tal como el de clases, de interacción secuencia, colaboración, actividades y estado. En esta sección sólo se presenta el diagrama de clases de un servicio, el de almacenamiento persistente; también se presenta algunos diagramas de secuencia en esta sección.

4.3.1 DIAGRAMA DE CLASES DEL SERVICIO DE ALMACENAMIENTO PERSISTENTE

En este modelo, el diagrama de clases define todos los atributos y métodos requeridos para la implementación del servicio. La Fig. 9 muestra el diagrama de clases del servicio de almacenamiento persistente. Se puede notar que las clases e interfaces sombreadas son aquellas que están relacionadas al servicio.

CloudServiceException contiene métodos para encontrar el tipo y causa de un error y debajo de él, las clases de error relacionadas al servicio de almacenamiento.

La interfaz de *PersistentStorageService* modela una forma de acceder y modificar el sistema de archivos, ya sea físico o virtual. Con él, el desarrollador puede listar las raíces del sistema (si existe más de una), consultar en su aplicación el espacio libre disponible y usado (dando un objeto persistente como referencia), crea y manipula archivos y directorios persistentes y abrir flujos de datos a los archivos.

Las interfaces de los servicios extienden de la interfaz *CloudService* que es una abstracción de cualquier servicio provisto por la nube. Esta clase incluye métodos para cubrir operaciones de consulta de estado del servicio, si es soportado por la nube, si el desarrollador tiene acceso y finalmente, para manejar el servicio. Note que el acceso del desarrollador a cada servicio es importante ya que la nube puede ofrecer distintos esquemas de acceso de acuerdo al plan de renta. Estos métodos deben ser comunes a todos los servicios.

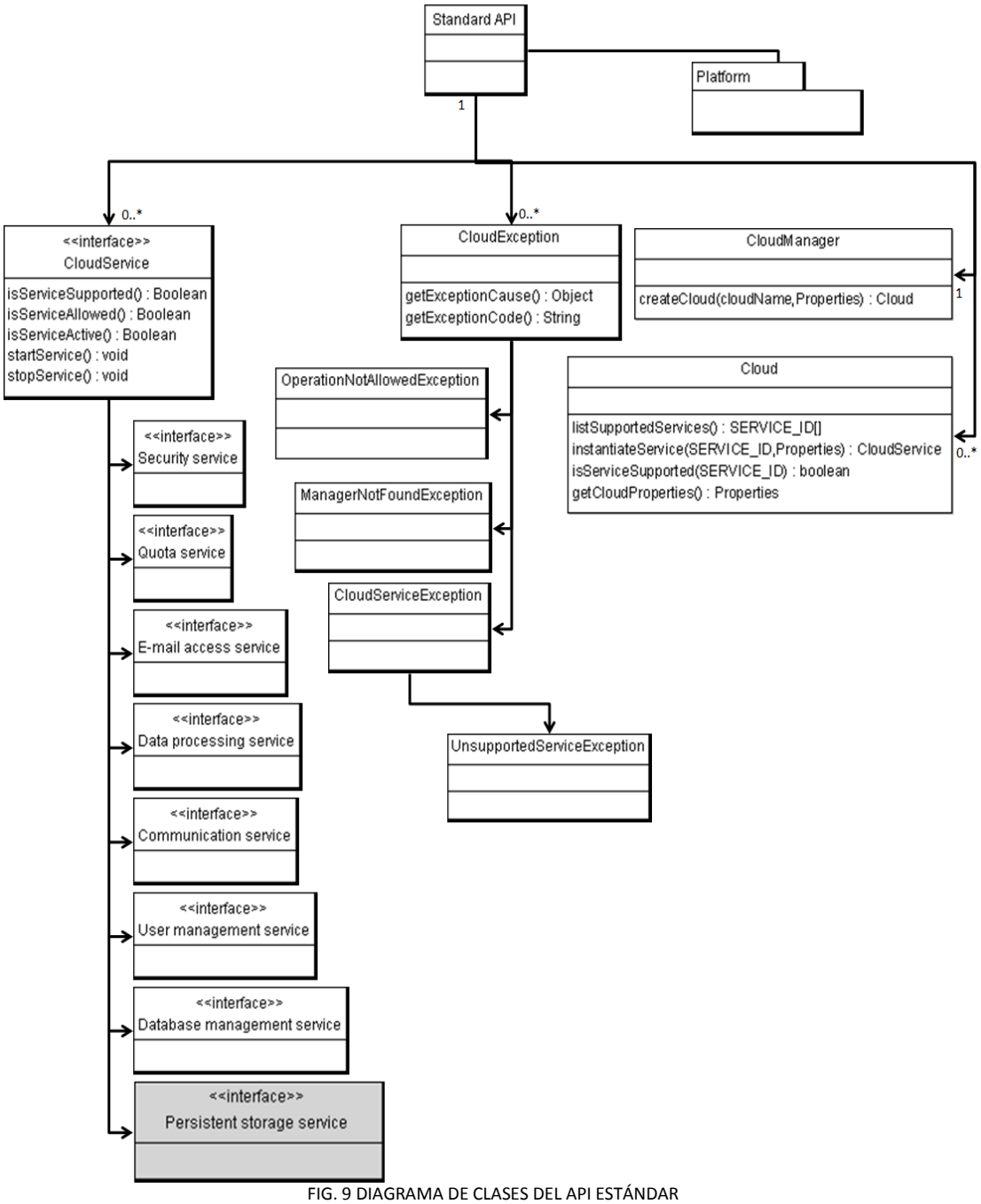


FIG. 9 DIAGRAMA DE CLASES DEL API ESTÁNDAR

4.3.2 CASOS DE USO PARA EL SERVICIO DE ALMACENAMIENTO PERSISTENTE

El servicio de almacenamiento persistente está modelado por la interfaz *PersistentStorageService*. Ésta define las funciones básicas que debe cumplir la implementación. Las funciones corresponden a los casos de uso del servicio.

Para comenzar a modelar el servicio hay que revisar los casos de uso que éste cubre. Cada uno de los casos de uso es una operación que un usuario realiza comúnmente al manejar archivos. Es necesario evaluar la forma en cada una de las nubes resuelve estos problemas, de forma que se pueda mostrar el proceso adecuado dentro de la clase final. La Fig. 10 presenta los casos de uso para el servicio de almacenamiento persistente.

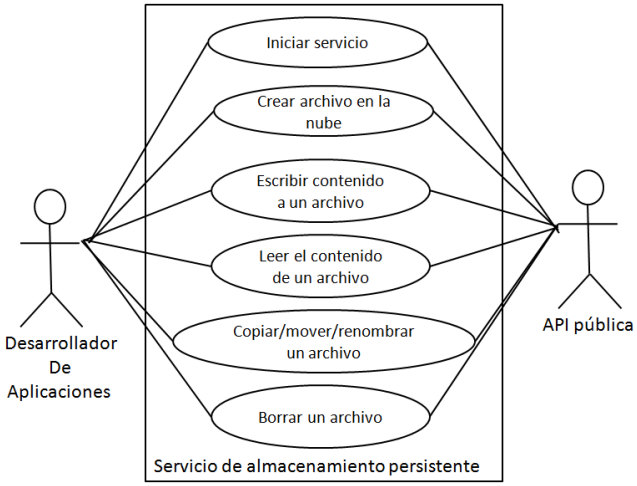


FIG. 10 DIAGRAMA DE CASOS DE USO PARA EL SERVICIO DE ALMACENAMIENTO PERSISTENTE

Posteriormente se presenta el diagrama de clases que modela específicamente el servicio de almacenamiento persistente. Este diagrama se basa en los casos de uso que se mostraron previamente. Este diagrama de clases se presenta en la Fig. 11. En la Fig. 12 se muestra el diagrama de composición de clases del servicio de almacenamiento persistente, con lo que se ilustran las clases de las que heredarán las diferentes instancias.

La interfaz *PersistentFSObject* describe un objeto que es almacenado en el sistema de archivos. Este objeto puede ser un *File* (que contiene datos), un *Directory* (agrupación de archivos) o un *Root* (raíz del sistema de archivos). Esta interfaz permite que se consulte información acerca del objeto, tal como su nombre, ruta, fecha de creación y modificación, permisos (estilo UNIX), atributos (por enmascaramiento de bits en un entero) y sobre su existencia. Las interfaces extendidas describen el objeto mencionado anteriormente y ofrecen tareas particulares relacionadas a cada uno. Finalmente se incluye la interfaz *PersistentFSObjectIterator* la cual modela un objeto que permite iterar sobre los objetos persistentes que contenga un directorio o una raíz.

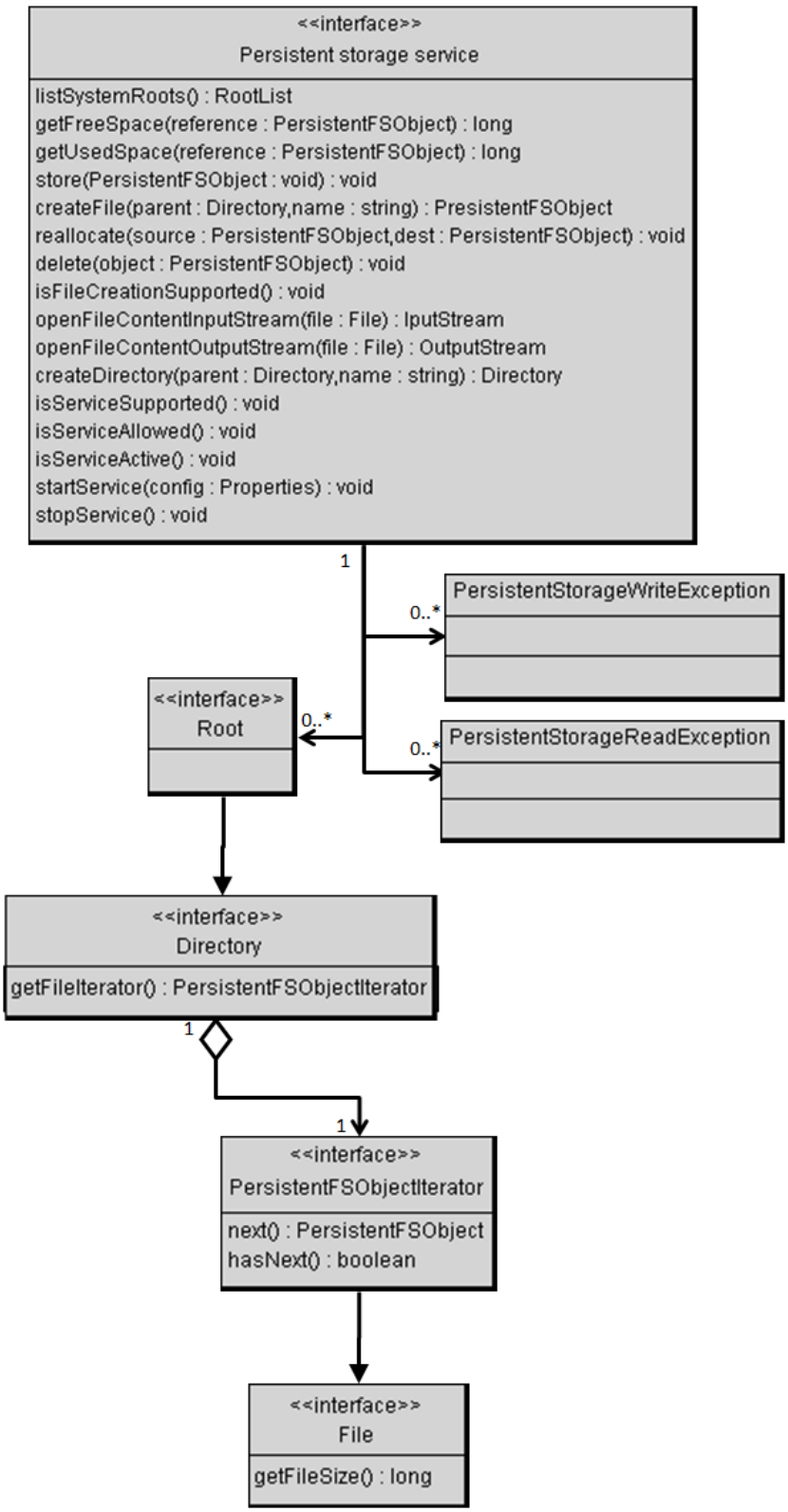


FIG. 11 DIAGRAMA DE CLASES DEL SERVICIO DE ALMACENAMIENTO PERSISTENTE

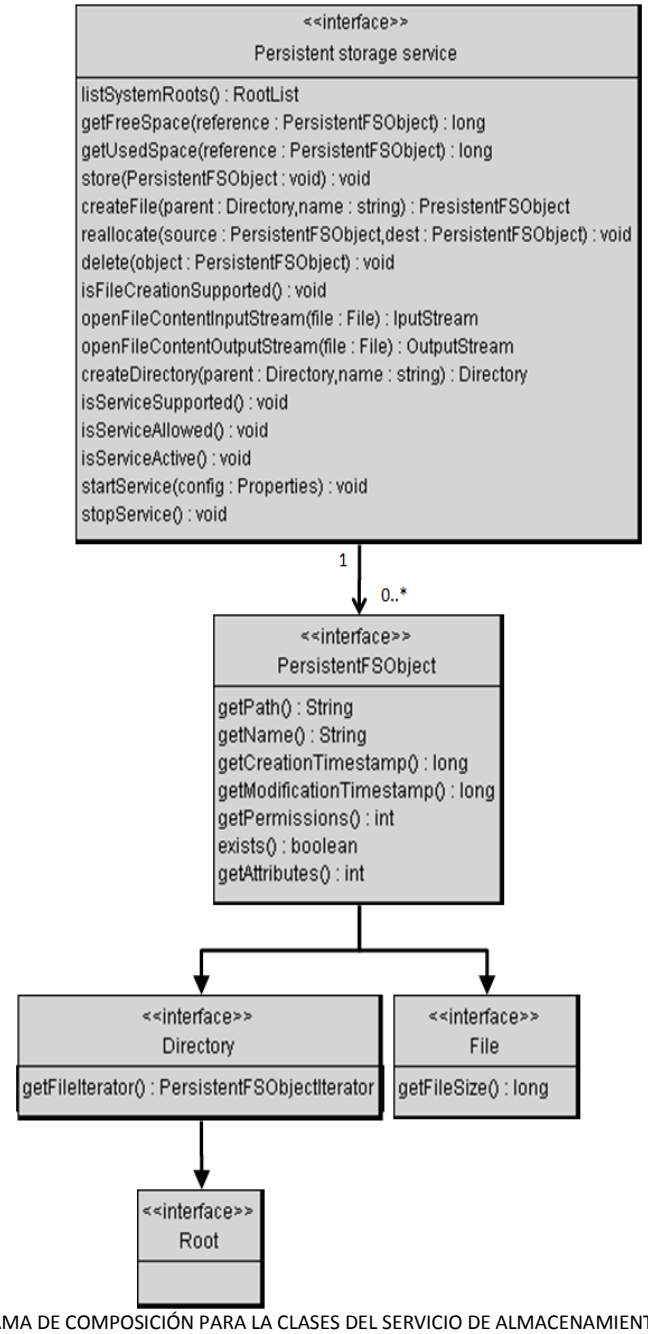


FIG. 12 DIAGRAMA DE COMPOSICIÓN PARA LA CLASES DEL SERVICIO DE ALMACENAMIENTO PERSISTENTE

4.3.3 DIAGRAMAS DE SECUENCIA

Debe haber un protocolo para desarrolladores a fin de usar un servicio. El diagrama de secuencia de la Fig. 13 muestra cómo cualquier servicio puede ser invocado mediante el llamado de métodos en cada interfaz de servicio. Con esta interacción el desarrollador está seguro de que el servicio estará disponible cuando la aplicación lo necesite o reaccione cuando no lo esté.

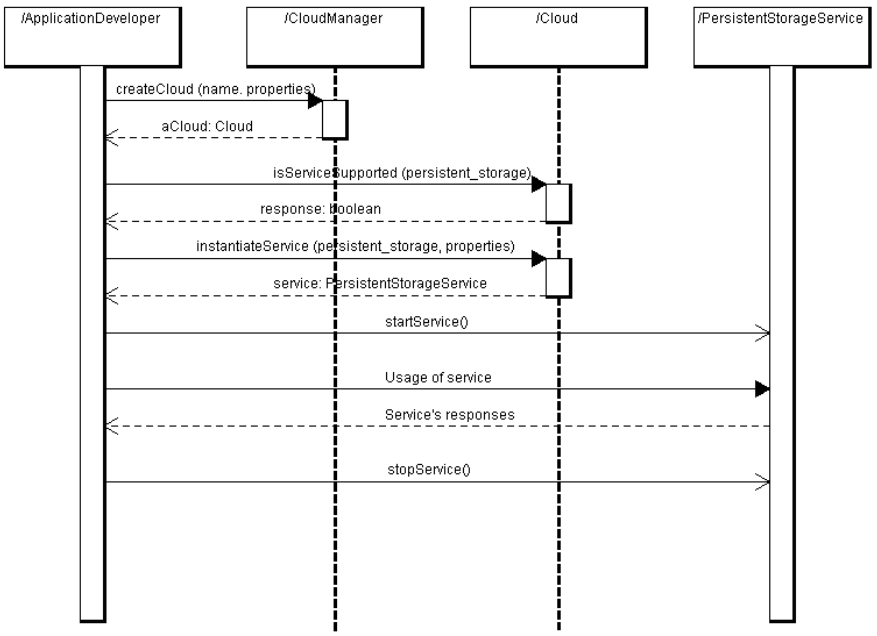


FIG. 13 DIAGRAMA DE SECUENCIA PARA LA INTERACCIÓN CON ALGÚN SERVICIO

El diagrama de secuencia mostrado en la Fig. 14 es incluido para ilustrar el proceso seguido por el desarrollador de la aplicación para lograr una tarea determinada. En este caso, para crear un archivo persistente en el sistema de archivos y consultar su tamaño.

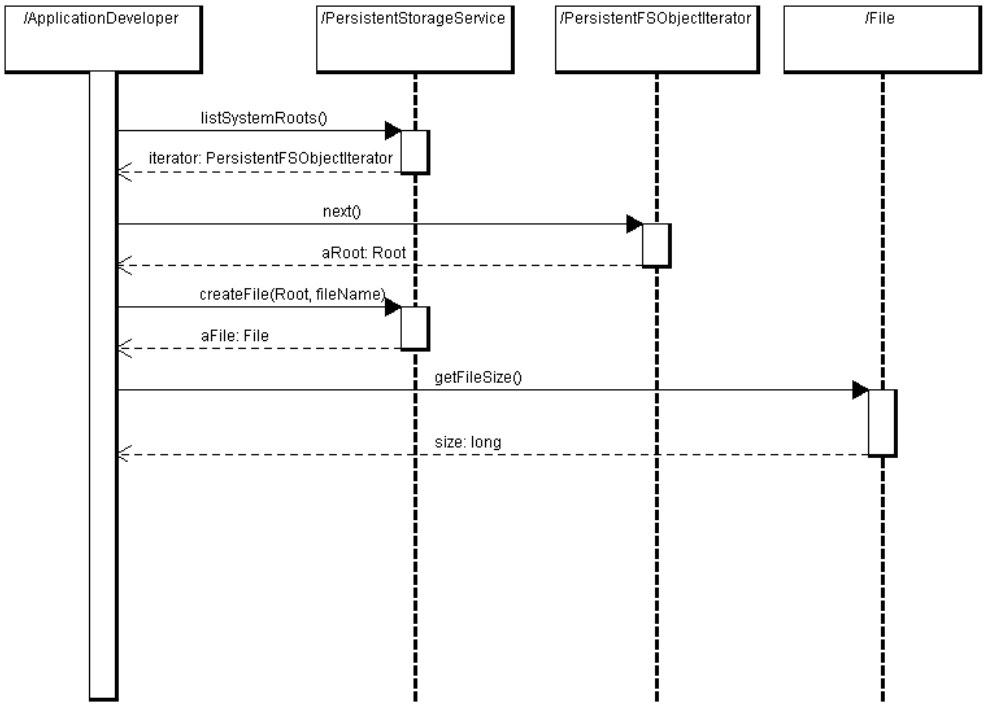


FIG. 14 DIAGRAMA DE SECUENCIA DEL SERVICIO DE ALMACENAMIENTO PERSISTENTE

4.4 MODELO DE IMPLEMENTACIÓN

Para comprobar que el API diseñada sea correcta y funcional, se realiza una implementación en Java a partir del modelo obtenido. En su mayor parte, el API contendrá clases abstractas y prototipos de métodos sin implementar. Eso es correcto ya que el desarrollador de aplicaciones sólo trata con invocaciones al API sin enterarse de su contenido. La funcionalidad completa la implementan los proveedores de las nubes.

En este caso, y sólo para fines experimentales, también se codificará el API para dos nubes particulares. Ya teniendo todo esto se podrán realizar pruebas controladas de lo modelado en una aplicación real.

4.4.1 IMPLEMENTACIÓN DEL API MEDIANTE CONECTORES

El API para cómputo en la nube que se está modelando es tan sólo el esqueleto o plantilla que deben seguir los proveedores de las nubes para modelar las bibliotecas que le dan al usuario para usar su infraestructura.

A esta parte abstracta que requiere ser implementada se le llama SPI (interfaz de proveedor de servicios) y es una de las dos facetas que presenta este proyecto. La segunda se mencionará durante el ejemplo funcional.

El resultado final de esta implementación será una biblioteca en Java que satisfaga los requerimientos del API modelada. A esta biblioteca le llamaremos *conector*.

El conector es hecho por el proveedor de la nube. Internamente, puede estar programado como mejor le convenga, pero es imperativo que cumpla con el contrato del API.

4.4.2 ARQUITECTURA DE IMPLEMENTACIÓN DEL API ESTÁNDAR

El API que se está modelando sigue una arquitectura basada en capas que se muestra en la Fig. 15. Estas capas son las siguientes:

Aplicación en la nube: La capa superior corresponde a la aplicación que crea un desarrollador para trabajar en la nube. Esta capa no se modela, pero se presenta en los diagramas como el desarrollador de aplicaciones. Dentro de la arquitectura de una nube, esta capa recae dentro de la capa de software (SaaS).

API estándar: El API estándar es lo que se está modelando. Para el desarrollador de aplicaciones no es más que una caja negra, pero internamente está compuesta por todas las clases que se presentaron en el modelo de diseño. Recae dentro de la capa de plataforma (PaaS).

Manejador de conectores: El manejador de conectores es una clase nativa del API estándar que no requiere ser implementada. Funciona como una capa de control que permite seleccionar qué implementación (conector) del API se usará para conectarse a una nube particular. Al igual que la anterior, recae dentro de la capa de plataforma (PaaS).

Implementación del API: Esta es la capa correspondiente al conector. Es la implementación que un proveedor de nube ha hecho del API estándar (en su faceta de SPI) para permitir el uso de su infraestructura. También entra en la capa de la nube PaaS.

API pública de la nube: El API pública es el conjunto de librerías que ya posee el proveedor para controlar su nube. Esta capa es opcional, pues un proveedor podría crear su conector de forma que trabaje directamente con su infraestructura. Está dentro de la capa PaaS de la nube.

Infraestructura de la nube: Es la nube, propiamente, comprende todos los recursos y servicios que ofrece el proveedor, que son requeridos por el desarrollador de aplicaciones y accedidos mediante las APIs. Esta capa del API está dentro de la capa IaaS de la nube.

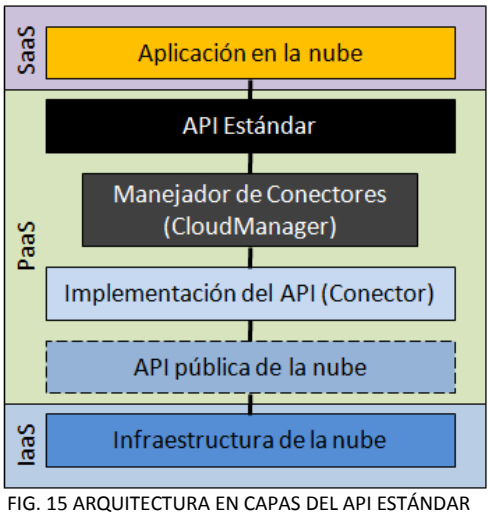


FIG. 15 ARQUITECTURA EN CAPAS DEL API ESTÁNDAR

Esta arquitectura en capas y conectores nos da flexibilidad para crear una aplicación que trabaje con múltiples nubes a la vez. Y permite que las implementaciones sean desarrolladas de acuerdo a las preferencias del proveedor.

4.5 CASOS DE IMPLEMENTACIÓN

Hay un punto importante a recalcar relacionado a la implementación que se está haciendo. Esto con la intención de dejar en claro que ésta tiene el único propósito de ejemplificar la forma en que se trabajaría con el API en las nubes y evaluar el modelo.

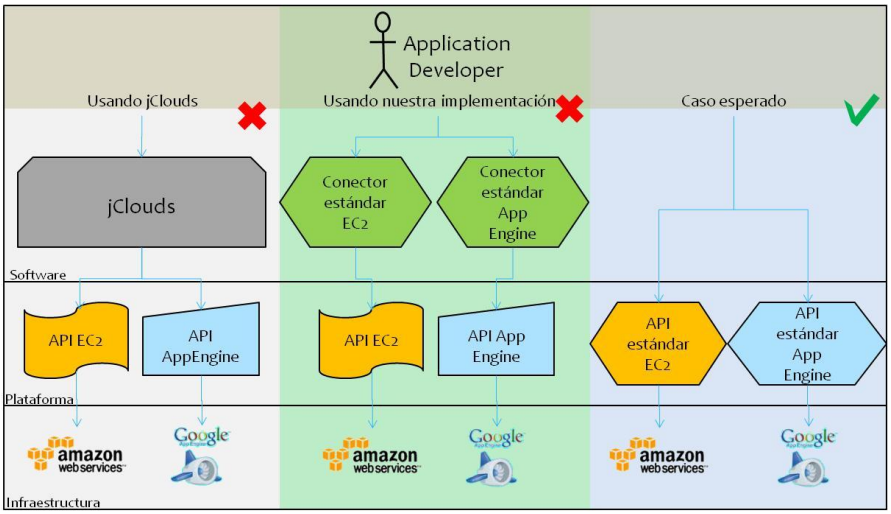


FIG. 16 CASOS DE IMPLEMENTACIÓN DE LA SPI

Cuando se analizó el estado del arte se mencionó una biblioteca llamada jClouds que trabaja como un intermediario entre las APIs de los proveedores y el desarrollador. Ésta biblioteca se coloca en la capa de software de la nube en la que se trabaja. Posee la enorme ventaja de brindar soporte para las nubes más populares de una forma unificada. Pero no es desarrollada por los proveedores de las nubes; es hecha por un tercero. Lo impráctico de ésta biblioteca es que, conforme salgan nuevas nubes al mercado, ésta necesitará nuevas versiones para soportarlas. Tarde o temprano la cantidad de nubes podría ser demasiado grande para ofrecer el soporte. Además de eso, constantemente los proveedores de nube liberan nuevos servicios para los que dan soporte a través de sus APIs públicas y si jClouds quiere soportarlas, deberá invertir en desarrollo constantemente. Por estas razones no parece recomendable tener una biblioteca centralizada de un tercero. En la Fig. 16 se muestra la forma en que se busca que el API estándar sea utilizado en cómputo en la nube.

En el caso de la implementación que se está haciendo para este trabajo, se hará una biblioteca por cada nube sobre la que se trabajará. Esta biblioteca también se coloca en la capa de software de la nube. El propósito de este proyecto no es que se hagan bibliotecas de este tipo por desarrolladores ajenos al proveedor de la nube. De hacerlo así se caería en algunos de los problemas que tiene jClouds; especialmente en el problema de soporte de nuevas funciones, pues la biblioteca intermediaria siempre estaría un paso atrás en las funciones que el API pública ofrece. La única razón por la que se está haciendo en este momento es para ejemplificar.

El caso deseado es que el modelo sea completo, funcional y con gran calidad. De ser así entonces se podría proponer a los proveedores de nubes que modelen sus APIs de

acuerdo con el contrato establecido por nuestra API. Así, el resultado sería una biblioteca implementada por el proveedor de la nube. Ésta se colocaría en la capa de plataforma, convirtiéndose en parte inherente de la nube. Las versiones de la biblioteca siempre estarían al día con la funcionalidad ofrecida por la nube. El usuario recibiría una biblioteca con la que trabajaría de forma estándar y podría migrar de una nube a otra con la confianza de que las herramientas que le ofrezcan en la nueva serán compatibles con sus sistemas ya desarrollados.

4.6 IMPLEMENTACIÓN DEL API ABSTRACTA

Usando el modelo de diseño, se construyo el modelo de implementación del API estándar en su parte abstracta, aquella que no implementa los servicios concretamente y sólo presenta las clases e interfaces que dan forma a los servicios.

Para la implementación se usó el ambiente de desarrollo (IDE) *Eclipse Indigo* junto con la versión 1.6 de *Java JDK*.

La implementación va de la mano con el modelo de diseño pero se anexaron y reorganizaron algunas clases con la finalidad de hacer el API más completo. De esta implementación se obtiene retroalimentación para mejorar el modelo y obtener una versión final adecuada más adelante.

En la Fig. 17 se presenta la jerarquía de clases resultante de la implementación.

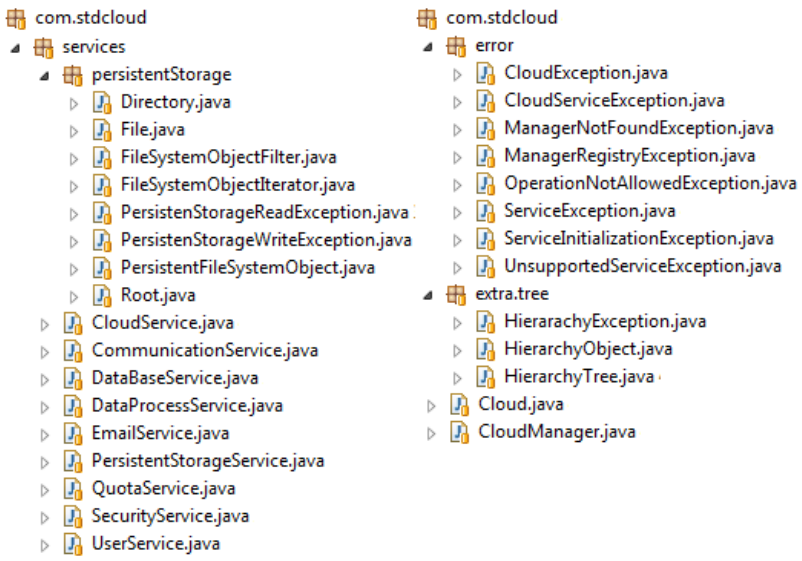


FIG. 17 ESTRUCTURA DE CLASES DEL API ESTÁNDAR

4.7 IMPLEMENTACIÓN DEL API PARA PRUEBAS

Ya se ha desarrollado una versión del API en su forma abstracta. Con esto pudimos presentar el modelo de una forma menos conceptual. Pero en este estado el API por sí sola no nos sirve para comprobar su funcionamiento. Es necesario ejecutar sus servicios en un entorno real y para ello se tendrán que implementar las clases que no contienen más que el esqueleto para llevarlo a un sistema completo.

Una vez que el API esté implementada, se podrá utilizar en una aplicación que haga uso de sus servicios. Con ello se podrá analizar y evaluar el API modelada.

4.7.1 IMPLEMENTACIÓN DE LA SPI (CONECTORES)

Para realizar la prueba del API estándar se crean dos conectores del API estándar. El primero corresponde a un conector para hacer uso del API pública de la nube de Amazon, mientras que el segundo trabaja con Google.

No se hará la implementación de la SPI completa; sólo se hará la parte central y en el servicio de almacenamiento persistente.

4.7.1.1 CONECTOR DE AMAZON

La nube de Amazon (EC2) ofrece un kit de desarrollo para acceder a sus servicios llamado Amazon AWS SDK, siendo la versión más actual la 1.3.7. Esta nube utiliza el concepto de máquinas virtuales por lo que permite el uso de aplicaciones de escritorio, Web. También abre sus servicios para ser accedidos fuera de su infraestructura. En cuanto a almacenamiento se refiere, maneja dos esquemas:

- Almacenamiento convencional dentro de sus máquinas virtuales, usando el sistema de archivos del sistema operativo instalado.
- Almacenamiento elástico, organizado en *buckets* (cubetas) donde se colocan archivos con una estructura plana pero con la posibilidad de una organización de directorios virtuales.

La documentación para el uso del kit de desarrollo de Amazon está disponible en línea para su consulta en [18].

4.7.1.2 CONECTOR DE GOOGLE

De forma similar a Amazon, Google provee un kit de desarrollo para el manejo de los servicios de su nube el AppEngine SDK. Su versión más reciente al momento es la 1.6.4. No se soportan aplicaciones de escritorio, únicamente Web y deben estar alojadas dentro

de la infraestructura de la nube. El SDK tiene conflictos con la versión 1.7 de Java, por lo que no se podrá trabajar con las mejoras que ésta ofrece.

El almacenamiento es elástico y también organizado en buckets. A los archivos se les puede asignar un id único (nombre del archivo) el cual puede presentarse como un subdirectorio, pero el almacenamiento realmente se hace en una estructura plana.

Al igual que Amazon, Google publica documentación para el uso adecuado de su kit de desarrollo en [19]. Durante el desarrollo de este proyecto, se noto que algunos casos de uso no están debidamente documentados y es necesario recurrir a blogs y grupos de discusión para solucionar ciertos problemas.

4.7.2 IMPLEMENTACIÓN DEL CORAZÓN DEL API

La primera parte que se debe implementar en el conector será la clase abstracta *com.stdcloud.Cloud*. El propósito de esta clase es el de encapsular una nube completa en un objeto que brinda servicios y provee información del sistema. Contiene una colección de identificadores de servicios (los 8 modelados) que se usan para crear instancias de cada uno. Estas instancias son de la clase *com.stdcloud.CloudService*.

La implementación de esta clase es en verdad simple. Tan sólo verifica que soporta el servicio que se le está pidiendo instanciar, crear internamente una instancia del servicio ya implementado y retornarlo en la petición. Para cada una de las nubes que se implementarán se hará una versión de esta clase, con lo que tendremos:

```
com.stdcloud.ec2.AmazonEC2Cloud – Nube de Amazon EC2
com.stdcloud.appengine.AppEngineCloud – Nube de Google AppEngine
```

Pero para que el usuario obtenga una instancia de alguna de estas dos sin pedirla por su nombre, habrá que implementar un método que las cree de forma segura.

El API para manejo de base de datos en Java (JDBC) ya ha resuelto este problema mediante una clase llama *DriverManager*, que se encarga de crear una conexión a una base de datos de acuerdo al driver (conector) instalado.

Siguiendo la idea de JDBC, se agregó la clase *com.stdcloud.CloudManager*, que tendrá una tarea similar. Vale la pena ver cómo se instancia una nube mediante el *CloudManager*.

En el código presentado en Fig. 18 se puede apreciar una llamada mediante *Reflection* de Java para obtener la clase *com.stdcloud.appengine.AppEngineManager* que corresponde al conector para la nube AppEngine de Google. Al hacer esta invocación, la clase *AppEngineManager* es cargada en memoria por el *ClassLoader* de Java. A su vez,

internamente, la clase *AppEngineManager* debe registrarse ante la clase genérica *CloudManager* usando su nombre como identificador.

```
try {
    Class.forName("com.stdcloud.appengine.AppEngineManager");
} catch (Exception e) {
    e.printStackTrace();
    return;
}
Cloud cloud = CloudManager.createCloud("AppEngineManager");
```

FIG. 18 INVOCACIÓN DE COUDMANAGER PARA UNA INSTANCIA DE APPENGINEMANAGER

Posteriormente el programador invoca el método estático *CloudManager.createCloud()* utilizando el nombre del manejador que necesita como parámetro.

Internamente, la clase *CloudManager* ya conoce a *AppEngineManager* y le hace una petición para crear una instancia de *Cloud*. Esta instancia será la implementación contenida en el conector. El resultado de la invocación será un objeto de la clase *AppEngineCloud* que es un *Cloud*.

Con este método, el programador puede diseñar sus aplicaciones de tal forma que cuando migre de una nube a otra no tenga que re-codificar y simplemente modifique un parámetro en su configuración.

4.7.3 CLASES EN LA IMPLEMENTACIÓN DE LA SPI

El resultado de la implementación fue un conjunto de clases Java que implementan la funcionalidad faltante del API. En la imagen siguiente se puede apreciar la organización de éstas en los paquetes correspondientes. Para fines prácticos, se uso el prefijo GAE y AppEngine para las clases del conector de Google, mientras que el de Amazon usa S3 y AmazonEC2. En la Fig. 19 Jerarquía de clases de los conectores para ambas nubeS

se pueden apreciar las clases que se implementaron para ambos conectores, nótese que no todas las clases del API se implementaron dado que algunas so auxiliares o no pertenecen a este servicio.

Se puede notar que la jerarquía para ambos conectores es la misma e incluso coinciden las clases implementadas. Esto se debe a que fue la misma persona la que realizó las dos implementaciones. De ser conectores desarrollados por los proveedores reales de las nubes, esto difícilmente ocurriría.

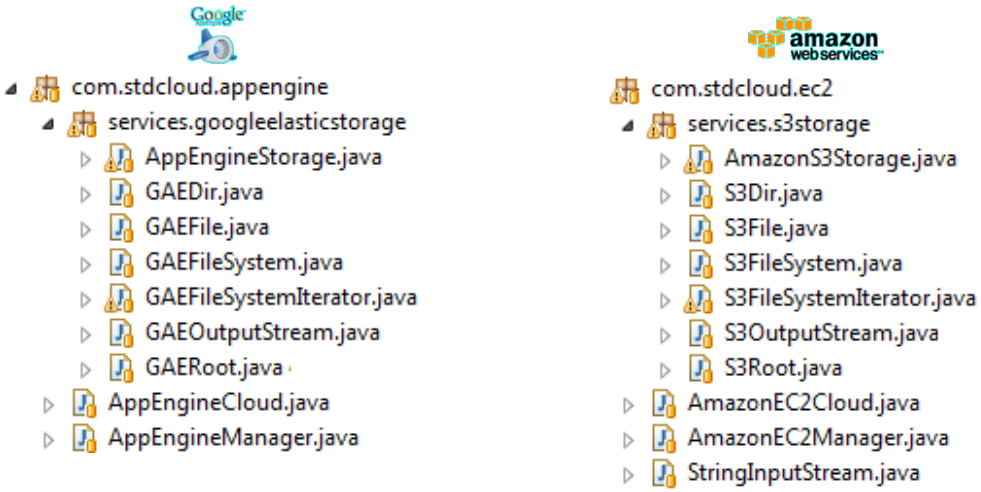


FIG. 19 JERARQUÍA DE CLASES DE LOS CONECTORES PARA AMBAS NUBES

Previo a la implementación de estos dos conectores, se hizo un conector más que trabajaba con el sistema de archivos del sistema operativo local. Sólo se implementó como referencia inicial y para facilitar las pruebas. No se entrará más a detalle sobre él dado que no es relevante para la evaluación.

4.8 IMPLEMENTACIÓN DEL API EN UN EJEMPLO REAL

Ya se ha trabajado sobre la implementación de la SPI. La cara restante del sistema que se está desarrollando es la de API, donde un desarrollador usa los servicios que permite acceder la biblioteca implementada.

Para probar el servicio de almacenamiento persistente del API se desarrolló una aplicación Web que hace uso de sus funciones. La aplicación es un manejador de archivos en la nube. Y soporta las siguientes funciones:

- Sobre directorios
  - Crear
  - Borrar
- Sobre archivos
  - Crear
  - Borrar
  - Renombrar
  - Copiar
  - Mover
  - Editar

La edición solo trabaja con texto, pero debe resultar suficiente para probar la lectura y escritura de archivos.

Para el desarrollo se utilizaron las siguientes herramientas:

- Eclipse JEE
- Java JDK 1.6
- Tomcat 6

En Fig. 20 se muestra la interfaz gráfica de la aplicación de ejemplo. Se utilizó la estructura que utilizan algunos manejadores de archivos populares para que el uso fuera familiar.

FIG. 20 INTERFAZ DE APLICACIÓN DE PRUEBA PARA EL API

La primera parte del desarrollo del manejador de archivos permitió probar el API en un desarrollo común. Brindó retroalimentación y mostró que el servicio se modeló con suficientes entradas y salidas para satisfacer las necesidades elementales de la aplicación. Este desarrollo se hizo con el conector de sistema de archivos local.

Como segunda parte se transportó a las nubes de Google y Amazon. Esto permitió probar que una aplicación que usa el API estándar puede ser transportada sin mayor complicación con solo cambiar el conector y realizando un ajuste no significativo.

En la Fig. 21 se presenta el código con el que se instancía el objeto Cloud y la inicialización del servicio de almacenamiento. Los cambios que se hicieron al código para migrar del conector local al de Amazon y al de Google se resaltan con recuadros.

```
private static void initCloud(Object anchor) throws Exception{
    try {
        /*Carga del manejador de la nube*/
        anchor.getClass().forName("com.stdcloud.ec2.AmazonEC2Manager");
    } catch (Exception e) {
        throw new Exception("Unable to load the CloudManager: " + e);
    }
    /*Carga de una nube*/
    currentCloud = CloudManager.createCloud("AmazonEC2Manager");

    /*Propiedades para arrancar instanciar el servicio de almacenamiento*/
    Properties storageServiceProperties = new Properties();
    storageServiceProperties.put("bucketName", "elasticbeanstalk-us-east-1-595592197131");
    storageServiceProperties.put("accessKey", "AKIAJJILZ2F4RZE7GIOQ");
    storageServiceProperties.put("secretKey", "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");

    storageService  = (PersistentStorageService)currentCloud.instantiateService(
            SERVICE_ID.PERSISTENTSTORAGE, storageServiceProperties);
}
```

```
private static void initCloud(Object anchor) throws Exception{
    try {
        /*Carga del manejador de la nube*/
        anchor.getClass().forName("com.stdcloud.appengine.AppEngineManager");
    } catch (Exception e) {
        throw new Exception("Unable to load the CloudManager: " + e);
    }
    /*Carga de una nube*/
    currentCloud = CloudManager.createCloud("AppEngineManager");

    /*Propiedades para arrancar instanciar el servicio de almacenamiento*/
    Properties storageServiceProperties = new Properties();
    storageServiceProperties.put("bucketName", "micky-gae-data");

    storageService  = (PersistentStorageService)currentCloud.instantiateService(
            SERVICE_ID.PERSISTENTSTORAGE, storageServiceProperties);
}
```

FIG. 21 VARIACIÓN DE LOS CÓDIGOS PARA TRABAJAR EN LAS NUBES DE AMAZON Y GOOGLE RESPECTIVAMENTE

Se puede ver en el código que la diferencia son variables de texto (el nombre de la clase del manejador, el nombre de la nube a instanciar y el conjunto de propiedades para inicializar el servicio). Esto simplifica la portabilidad dado que se podrían llevar a archivos de configuración y dejar sin modificaciones el código.

El resultado de la migración de la aplicación a ambas nubes fue altamente satisfactorio. El cambio en el caso más extenso (Amazon) fue de 5 líneas. Considerando las 1170 líneas de código que conforman la aplicación completa (incluyendo HTML) tan sólo representa un 0.4%. Esto prueba, en este experimento local, que el API cumple con su propósito debidamente.

4.9 EVALUACIÓN DEL API ESTÁNDAR

El resultado a obtener en este proyecto es un API que facilite el desarrollo de aplicaciones en la nube. Para lograr esto, es muy importante que el diseño tenga calidad.

Un API es software y el proceso para su desarrollo es similar al de cualquier otro sistema. Pero al ir dirigida a desarrolladores y no a usuarios finales, se deben evaluar algunos criterios específicos. Esto no será del todo fácil, ya que ciertos aspectos son muy abstractos y subjetivos.

Un API debe ser elegante y atractiva a primera vista. Esto para ganar el interés de los usuarios desde un inicio. Atractiva refiriéndose a que el API sea clara y sencilla, sin dejar de ser robusta. Elegante al lograr que los procedimientos simples sean fáciles. Pero estos puntos no lo son todo.

Los siguientes son algunos criterios para evaluar la calidad de software.

El contrato API – Usuario: La parte pública del API (los métodos, clases, atributos, etc. que el desarrollador puede acceder y utilizar) se conoce como la firma de métodos y campos. Esta firma es un contrato que se genera entre el API y sus usuarios desde su primera liberación. Si la firma cambiara, los programas donde los usuarios la utilizaron podrían dejar de funcionar.

Esto afecta a la etapa de mantenimiento del API, donde es muy importante respetar el contrato establecido.

En el caso del API estándar para cómputo en la nube, el diagrama de clases presenta las entidades que son visibles al usuario. Siendo esta la primera versión, define la firma del API.

Plan de evolución del API: Los usuarios de un sistema de software siempre buscarán más de lo que ya se les dio y las APIs no son la excepción. Si un API tiene potencial para crecer en versiones posteriores a la liberación, hay que prevenir y planear este crecimiento. Se debe prever cómo podría evolucionar naturalmente el API y hacia a dónde, a partir de lo que se está diseñando inicialmente. Para ello se debe tener un plan de evolución que indique cómo se puede incluir nueva funcionalidad o módulos para hacer una nueva versión del API, sin que ésta pierda su compatibilidad con las anteriores.

En el caso del API estándar, esta evolución se planeó pensando en que los servicios de las nubes en el futuro pueden aumentar (difícilmente se reducirían). Debido a esto, la existencia de un servicio no es dependiente de otro, ya que se manejan con interfaces separadas. Y la instanciación de los servicios se hace mediante la llamada a una función general (*instantiateService*) cuyo parámetro es el identificador del servicio deseado (*PERSISTENTSTORAGE*) perteneciente a una enumeración (*SERVICE_ID*) y un conjunto de propiedades variables (*storageServiceProperties*), como se puede ver en el código de la Fig. 22.

```
Properties storageServiceProperties = new Properties();
storageServiceProperties.put("bucketName", "gae-data");
PersistentStorageService googleStorageService  =
        (PersistentStorageService) googleCloud.instantiateService(SERVICE_ID.PERSISTENTSTORAGE, storageServiceProperties);
```

FIG. 22 CÓDIGO FUENTE PARA INSTANCIAR EL SERVICIO DE ALMACENAMIENTO PERSISTENE EN UNA NUBE

Si en futuro se tuviera un servicio en las nubes que controlara, por ejemplo, funciones de video, una nueva versión del API se liberaría con una nueva interfaz llamada *VideoService* y la enumeración *SERVICE_ID* incluiría un identificador para éste. Cuando se liberan nuevas versiones de un API es fácil agregar cosas, pero nunca quitar.

4.9.1 RESULTADOS DE LA EVALUACIÓN

De acuerdo a lo definido por Jaroslav Tulach en su libro “Diseño práctico de APIs”, se toman seis criterios para comprobar la calidad del API, haciendo un análisis de lo ya diseñado. [20] Utilizando como base la implementación de los conectores para Google y Amazon podemos caracterizar una valoración de la SPI así como utilizar el ejemplo del manejador de archivos para calificar el API.

***Organización del API a distintas audiencias**:* El API que se está diseñando en este trabajo también tiene características de SPI, por lo que la evaluación se debe hacer desde dos puntos de vista:

Quien usa el API (Interfaz de programación de aplicaciones): Es el punto de vista de un usuario programador que toma el API como caja negra y usa los servicios que ésta ofrece para crear una aplicación en la nube. Al hacer referencia a este ángulo hablaremos del API.

Quien implementa la SPI (Interfaz de proveedor de servicios): Es la visión que tienen los proveedores de nube que crean una biblioteca, acorde a su plataforma. Remueven la abstracción del API implementando los métodos, clases y procedimientos faltantes. Al hacer referencia a este ángulo hablaremos de la SPI.

Los dos puntos de vista (el de API y el de SPI) provocan, de forma natural, una división del API estándar, que al estar bien organizada, evita confusión en los usuarios. Todos los servicios que ofrece la SPI deben ser implementados por el proveedor de la nube. A su vez, esos mismos servicios tienen acceso para el usuario mediante el API.

Suficiencia: Hacia el usuario, el API estándar cubre las necesidades básicas para las que fue diseñada y lo debe hacer correctamente. Hacia el implementador, la SPI tiene flexibilidad suficiente para que él pueda integrar los servicios que ofrece. Podemos recalcar estos puntos en el modelo del API estándar:

- Las clases para implementar son públicas y no finales.
- Cubre todos los casos de uso del servicio implementado, como se muestra en el diseño del servicio del almacenamiento persistente.
- Las funciones para el manejo del servicio son públicas.
- Contiene objetos auxiliares que no requieren implementación, como las excepciones y el manejador de nubes.
- Utiliza parametrización flexible en la instanciación de nubes y de servicios, como el uso del objeto *Properties* de java.
- Cada servicio se modeló en una clase separada, por lo que no se entorpecen entre sí, lo que se puede apreciar en el diagrama de clases del API estándar del modelo; a su vez, esto permite que se incorporen nuevos servicios en el futuro.

Comprensibilidad: Si el usuario del API o el implementador de la SPI no se entienden con ella, puede ser que no se haya estudiado correctamente el conocimiento previo de los usuarios a los que fue dirigida. Se tiene que estimar cuál es el perfil promedio o básico con el que se puede trabajar. Al preocuparse por esto se busca que el API sea fácil de

comprender sin sacrificar la suficiencia ni el desempeño. El perfil buscado en los usuarios del API estándar es el de programador de nivel medio. No es necesario un conocimiento avanzado para explotar la funcionalidad.

Al implementador de la SPI se le debe dar un paquete muy fácil de entender; de lo que se pretende hacer y el cómo, pues una mala interpretación de su parte tendrá resultados impredecibles.

El proveedor de nube necesita hacer la implementación del API para su plataforma y él debe entender perfectamente cuáles son los requerimientos y los resultados que se espera obtener de cada clase y cada método. Si no es así, el API de una nube no responderá de la misma forma que la de otra y se perderá la portabilidad nuevamente.

Estos puntos se siguieron en el modelado del API estándar y vale la pena recalcarlos (todos estos se pueden ver en la definición de clases del modelo de diseño):

- Usa terminología clara y nomenclaturas estándar. [21]
- El API estándar está escrita en inglés.
- No requiere técnicas avanzadas de programación.
- Los nombres de clases y métodos son descriptivos.

Consistencia: La consistencia en una API abarca desde la forma en que se nombra a las entidades que la conforman hasta sus versiones.

Los tipos de dato que se escogieron para ciertos casos se están usando en todos los lugares donde la aplicación es la misma, de forma que el usuario pueda recordarlo y después predecir lo que se usará en métodos que aún no conoce.

Ya se habló del contrato entre el usuario y el API. Éste debe prevalecer a través de la evolución del sistema; proveer compatibilidad hacia versiones anteriores sin contradecir estructura ni funcionalidad.

En el modelo del API estándar se puede apreciar la consistencia desde el nombre de las clases. Por ejemplo, el nombre todas las interfaces de servicios tiene el prefijo *Service*, al igual que las clases referentes a excepciones tienen en su nombre el prefijo *Exception*.

Por otro lado, el proceso de creación de instancias de nubes y de servicios requiere parámetros para su inicialización y estos se mantienen consistentes en los tipos de datos que recibe (*Properties*).

Exhibición: La exhibición se refiere a la facilidad que da el API al usuario para que la entienda y aprenda. Tanto del lado del API como de la SPI se debe proveer información clara y completa de cómo se utiliza el sistema. Estos son algunos puntos de la evaluación de exhibición:

- El API abstracta cuenta con documentación en código (*Javadoc*).
- El modelo UML desarrollado se puede utilizar como referencia.

***Preservar la inversión de los usuarios*:** Un desarrollador invierte tiempo y esfuerzo al crear una aplicación y si ésta utiliza APIs de terceros, necesita que esa inversión no se pierda a futuro. Re-trabajar una aplicación por culpa de un componente externo sólo causará que el desarrollador opte por buscar mejores y más consientes herramientas.

Este punto es particularmente importante en el diseño del API estándar dado que la audiencia a la que va orientada está compuesta por usuarios programadores que no pueden darse el lujo de estar arreglando sus aplicaciones cada vez que el API cambie. A su vez, si por el lado de la SPI, no se respetara el contrato, provocaría que los proveedores de ajusten bibliotecas y tendrían un periodo de fallas en un porcentaje alto de sus aplicaciones soportadas. No se ha liberado ninguna versión por lo que es muy pronto para evaluar este punto.

4.10 Comparativa del API estándar con Java Data Base Connectivity (JDBC)

Una forma más de comprobar la calidad del API estándar es compararla con otra que ya es conocida por ser bien diseñada. Algunos de los manejadores de bases de datos para los que se han hecho implementaciones de JDBC son MySQL, MSSQL, Oracle, MS Access, entre otras.

Java Data Base Connectivity (JDBC) es el API utilizada por los desarrolladores de aplicaciones en java para realizar conexiones y manipulación de bases de datos. Es un paquete muy robusto y estable que cuenta con tres características principales [11]:

- Establecer conexiones a bases de datos o cualquier origen de datos tabular
- Enviar sentencias SQL
- Procesar los resultados

Pese a que pareciera ser muy limitada en su funcionamiento, con estos tres módulos un desarrollador consigue el control total de una base de datos en una aplicación Java.

Al igual que el API estándar para cómputo en la nube que se está diseñando, JDBC requiere la implementación de la parte del API correspondiente a la SPI, con lo que se consigue tener drivers. La arquitectura de JDBC se basa en capas. Como se muestra en la Fig. 23.

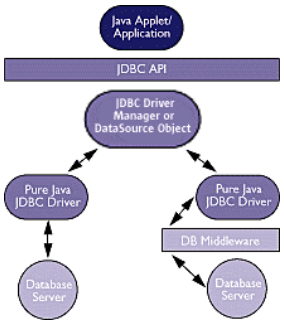


FIG. 23 ARQUITECTURA DEL API JDBC

La capa superior es la aplicación Java en la que se necesita manejo de base de datos. La capa siguiente es el API de JDBC que se maneja como una caja negra para los desarrolladores. Posteriormente se tiene el manejador de drivers, que permite realizar la conexión a más de un tipo de bases de datos y utilizar en paralelo varios drivers. Más abajo nos encontramos con la implementación de la SPI, que es el driver. En la rama de la derecha de la arquitectura, se puede identificar una capa correspondiente a middleware, que no es más que un complemento necesario para conectar el driver con la base de datos real. Finalmente se tiene el servidor de base de datos.

Comparado con la arquitectura del API estándar para cómputo en la nube presentada en la Fig. 15, se puede apreciar que el cambio es únicamente en el recurso final: en JDBC es una base de datos y en el API estándar es la infraestructura de una nube.

4.10.1 COMPARATIVA DE CALIDAD ENTRE JDBC Y EL API ESTÁNDAR

Para comparar la calidad del API estándar se toman todos los puntos mencionados en la evaluación y se aplican a ambas APIs. Esta comparativa se presenta en la Tabla 2.

TABLA 2 TABLA COMPARATIVA DE CALIDAD ENTRE JDBC Y EL API ESTÁNDAR

Criterio de comparación	JDBC	API estándar	Resultado
Contrato con el usuario	Quedo definido como la parte pública del API. Este contrato se puede ver en la documentación Javadoc de JDBC disponible en su sitio oficial.[11]	Igualmente definido con la parte pública del API. Este contrato se puede consultar en la documentación Javadoc del API o en los diagramas UML del modelo de diseño.	**Superior**
Plan de evolución	El plan de evolución con el	La inclusión de servicios	**Bueno**

	que se modeló JDBC es desconocido, pero el modelo de base de datos relacional no cambia, por lo que la estructura que maneja podría no necesitaría alteraciones.	aún no existentes en las nubes se logra al tener la interfaz de cada servicio independiente de las otras y permitiendo la instanciación mediante un método genérico.	
Organización del API a distintas audiencias	El API JDBC ofrece un paquete nativo de java de bajo nivel para realizar drivers eficientes.	No se tiene un paquete adicional para la SPI, se comparte con el API, aunque las clases están organizadas adecuadamente, no es tan claro como en JDBC	**Peor**
Suficiencia	Cumple con las características esenciales con las que se presenta. Las clases del contrato son interfaces y clases públicas no finales (como la clase *Connection*). Contiene objetos auxiliares que no requieren implementación (como el objeto para fechas *Timestamp*). Permite hacer consultas mediante sentencias de texto SQL o creándolas de forma dinámica. Permite trabajar con más de una base de datos a la vez gracias a su manejador de drivers.	Cumple con todos los casos de uso que cubre el modelo de análisis. Igualmente, las clases del contrato son interfaces y clases públicas no finales (como la clase *EmailService*). Usa objetos auxiliares sin implementación, como las excepciones (*CloudException*). Permite la instanciación de servicios de forma genérica. Permite trabajar con más de un conector de nube a la vez.	**Bueno**
Comprensibilidad	Usa terminología estándar basada en Java Beans, como al obtener el atributo entero de alguna tupla en el resultado de una consulta mediante la función *getInteger()* de la clase *ResultSet* que usa el prefijo estándar "get". Está escrita totalmente en inglés. Los nombres de los objetos son muy claros a su propósito: una conexión se llama *Connection*, una sentencia se llama *Statement*, un conjunto de resultados se llama *ResultSet*. Esto también se	Usa terminología estándar de acuerdo a Java Beans, como en los métodos para preguntar si algo ocurre, que usan el prefijo "*is*" como *isServiceSupported()* de la interfaz *ClooudService*. Así como el prefijo "get" en las funciones para obtener las propiedades de un *PersistenFSObject*. Está totalmente escrita en inglés. Los nombres de los objetos son descriptivos, como por ejemplo la clase	**Bueno**

	cumple con sus métodos: si se quiere crear una sentencia se llama a *createStatement()*. Su funcionamiento se hace intuitivo, gracias a esto.	perteneciente a un Directorio se llama *Directory* y la interfaz del servicio de cuota se llama *QuotaService*. El uso del API se facilita al tener estos puntos	
Consistencia	Todas las funciones con propósitos similares se invocan de la misma forma y contienen nombres similares, como al solicitar un dato entero de un *ResultSet*, se le pide directamente mediante el método *getInteger*() y si se quiere un flotante se obtiene con *getFloat()*. Al ejecutar una consulta se llama a *executeQuery()* y al ejecutar una modificación es *executeUpdate().* En cuanto a procedimientos, para usar una conexión, esta se abre, se usa y se cierra. Y para usar el resultado de una consulta esta se ejecuta, se usa y se cierra.	De igual forma, las funciones que tiene un uso similar conservan la misma nomenclatura en sus nombres. Para pedir el espacio libre del sistema de archivos, se llama al método *getFreeSpace()* mientras que obtener el usado es mediante *getUsedSpace().* La parametrización también es consistente. Tanto para abrir un flujo de escritura como de lectura, el parámetro requerido es un *File* del archivo que se escribirá.	**Bueno**
Exhibición	Cuenta con Javadoc completo y manuales de uso en su sitio oficial. No cuenta con documentación UML oficial.	Cuenta con modelo UML y Javadoc. Falta crear manuales de uso.	**Bueno**
Preservar la inversión de los usuarios	Actualmente JDBC se encuentra en la versión 4.0 que puede reemplazar sin conflictos a cualquier otra versión anterior. Cada versión nueva ha ido incluyendo mejoras, por ejemplo: de la versión 1 a la 1.2 se incorporó que los *ResultSet* pudieran ser modificados. De la versión 3 a la 4 se incluyó que los drivers se cargaran automáticamente.	No se ha liberado ninguna versión para evaluar este punto, lo único que se puede considerar es que el contrato con el usuario se respetará.	**No aplicable**

En conclusión de la evaluación, el API estándar desarrollada es un comienzo estable para modelar el API completa para cómputo en la nube. Dos características fueron inferiores que JDBC; una es inevitable debido que aún no se libera ninguna versión y la otra debido a que la organización del API JDBC incluyó en una versión posterior a la inicial un módulo nativo de java para la implementación de la SPI.

Es importante que esta comparación se tome como base para el resto del modelado y que no se caiga en vicios de programación ni en incompatibilidades dentro de la misma API.

5 CONCLUSIONES

Combinando la teoría de sistemas distribuidos y la transparencia de recursos que ofrece la virtualización, el cómputo en la nube brinda una excelente y novedosa plataforma para el desarrollo de aplicaciones y su liberación mediante Internet.

El cómputo en la nube está comenzando a adoptar estándares en algunos rubros clave que permiten harmonía al momento de utilizarla. Estándares para formatos de virtualización, datos y arquitectura permiten la migración de sistemas operativos completos, interoperabilidad y una mejor servicio hacia el usuario (tanto programadores como usuarios finales). En cuestión de portabilidad de aplicaciones, aún se tiene un terreno parcialmente explorado.

Cada día más desarrolladores y empresas prefieren liberar sus sistemas de software al público usando alguna nube debido a las ventajas de costo y escalabilidad que se les ofrece, sin olvidar a la comunidad científica que la elige usar como herramienta de procesamiento paralelo. A su vez, los proveedores de nueves liberan nueva funcionalidad para sus plataformas frecuentemente. Este crecimiento debe ser guiado, sin limitarlo; llevarlo de forma natural por un camino accesible para los usuarios y que pueda ser adoptado por todos los proveedores sin perder su unicidad a la hora de competir en el mercado.

5.1 QUÉ SE HIZO Y CÓMO

En este trabajo se identificó la problemática que se tiene cuando una aplicación creada para explotar los servicios de una nube es migrada (portada) a otra. Se experimentó realizando una aplicación Java sencilla para el manejo de archivos en la nube de Google AppEngine y se intentó migrar a la de Amazon AWS en sus dos vertientes:

almacenamiento convencional y elástico. Para ello se utilizaron los kits de desarrollo estándar que ofrecen estos dos proveedores. Al hacer esto se comprobó que existe una dependencia entre el código de la aplicación con la nube sobre la que se trabajó inicialmente. Esta dependencia se percibe a primera vista en los encabezados de las clases creadas, donde el nombre del proveedor de la nube aparece constantemente y denota que no se está trabajando de forma genérica. Pero va más allá; en cuestión del protocolo a seguir para realizar una misma tarea, se tienen diferencias clave que evitan que se puedan mover las aplicaciones de una nube a otra: como los tipos de estructuras que se usan y su inicialización.

Una vez que se comprobó el problema, se analizó el estado del arte para identificar las posibles soluciones. Durante este proceso se vio que algunos organismos internacionales (como el NIST y el IEEE) ya han mencionado el problema y propuesto que se trabaje sobre él sin dar una solución concreta. Una forma de resolver este problema de portabilidad emerge con una perspectiva ambiciosa. JClouds es un API no estándar desarrollada por una empresa privada que se dio a la tarea soportar las nubes comerciales populares. Se posiciona como una capa entre la aplicación del desarrollador y el API pública de la nube y brinda una metodología unificada para trabajar sin importar el proveedor. Dicha propuesta es la única forma de realizar portabilidad efectiva actualmente pero, aunque es una solución funcional, no es la mejor aproximación, dado que el crecimiento del mercado del cómputo en la nube superará las capacidades para dar soporte en las implementaciones de jClouds.

La propuesta que se tiene consiste en buscar un estándar internacional para el desarrollo de APIs de cómputo en la nube. Para lograr esto, se diseña el modelo de un API estándar que comprende los servicios básicos ofrecidos por los proveedores de nubes y se integran en una metodología de uso homogénea. Este modelo se crea basándose en el Proceso Unificado de Desarrollo de Software y auxiliándose en el Lenguaje Unificado de Modelado.

El API diseñada es abstracta, lo que implica que tan sólo es el esqueleto de una librería completa. Define las entradas y salidas en los procesos, los tipos de datos a utilizarse y el protocolo a usar al interactuar con los diversos servicios. Vale la pena destacar que la abstracción del API debe ser removida por los mismos proveedores de las nubes, de forma que la implementación sea eficiente, completa y correcta. A la liberaría resultante de quitar la abstracción se le denomina “conector” Esto también permite que el desarrollador de aplicaciones tenga la seguridad de que trabaja con conectores que están al día con la funcionalidad de la plataforma sobre la que se encuentra.

Para proponer el API estándar se realizó el modelo de análisis, donde se identificaron los dos actores que interactúan en el sistema (el desarrollador de aplicaciones y el API pública de la nube) y como casos de uso se presentan los ocho servicios básicos (almacenamiento persistente, base de datos, manejo de usuarios, e-mail, cuota, canales de comunicación, seguridad y procesamiento de datos); el modelo de diseño, donde se detallan las clases y se introducen los métodos que estas tienen, se profundiza en el servicio de almacenamiento persistente y se muestran algunos diagramas de secuencia para ilustrar el protocolo estándar que se pretende usar; el modelo de implementación donde los diagramas se llevan a código fuente, creando las clases abstractas e interfaces que forman el esqueleto.

Ya teniendo el modelo hecho, se continuó con una fase de pruebas, donde se removió la abstracción de la librería creando conectores, tal como se pretende que lo hagan los proveedores de las nubes. Se desarrolló una aplicación para el manejo de archivos en la nube (derivado de las pruebas originales) que trabaja con el API estándar y se llevó de un sistema local a la nube de Google y de ahí a la nube de Amazon con éxito y facilidad.

Finalmente se presenta una etapa de evaluación de la calidad del API para comprobar que lo que se está haciendo es correcto y suficiente. Para hacerlo, se evalúan dos puntos importantes: el contrato del API con el usuario y el plan de evolución. Y luego se analizan seis rubros de evaluación que, de una forma subjetiva, se califican para dar un veredicto de la calidad del modelo hecho. Estos son: suficiencia, comprensibilidad, exhibición, consistencia, organización para diferentes audiencia y preservar la inversión del usuario. Dado que es difícil hacer esta evaluación de forma objetiva, se toma como punto de comparación el API para manejo de bases de datos JDBC, la cual se evalúa bajo los mismos criterios y se muestra una tabla comparativa con los resultados de ambas APIs.

5.2 CÓMO MEJORAR LO HECHO Y TRABAJO FUTURO

Este trabajo pretende ser el inicio de un modelo completo que pueda ser propuesto ante los organismos de estandarización internacional y ser adoptado por los proveedores de las nubes. Aún es necesario trabajar sobre él, ya que sólo se detalló el servicio de almacenamiento persistente. Se propusieron siete servicios más que no han sido modelados y se sabe que existe otro servicio más que debe ser incluido (servicio de balance de carga). Y así como aparece este servicio, es posible que surjan otros que necesiten ser integrados. El API está diseñada para soportar este crecimiento, sólo es cuestión de darse a la tarea de terminar el modelado.

Otro punto importante, sobre el que se debe trabajar, es la evaluación de calidad. Para hacerla adecuadamente, es necesario hacer rondas de prueba con un grupo de desarrolladores que implementen algún sistema sencillo en la nube haciendo uso del API estándar. Después de esto se pueden realizar encuestas que demuestren de forma estadística qué tan elegante, atractiva y buena es el API.

Finalmente, al tener el modelo completo, se deben los resultados y usarlos como base para proponerlo ante la comunidad científica y los organismos de estandarización. Claramente se busca que este modelo sea tomado como el estándar internacional para la creación de APIs para el cómputo en la nube pero, a fin de beneficiar a la comunidad de desarrolladores de cómputo en la nube, basta con iniciar el movimiento que incite a otras personas trabajando en la misma área a realizar sus propias propuestas, participar en el concurso y lograr que pronto se logre la estandarización.

6 BIBLIOGRAFÍA

[1] Booch, Jacobson y Rumbaugh. "El Proceso Unificado de Desarrollo de Software". Ed. Addison Wesley, Madrid, 2000.

[2] Booch, Jacobson y Rumbaugh. "El Lenguaje Unificado de Modelado". Ed. Addison Wesley, Masschusetts, 1999.

[3] "IBM Introduces Ready-to-Use Cloud Computing". Disponible en: http://www-03.ibm.com/press/us/en/pressrelease/22613.wss

[4] Peter Mell y Timothy Grance. "The NIST definition of cloud computing". Publicación especial del NIST 800-145. 2011.

[5] BhaskarPrasad Rimal, Eunmi Choi, Ian Lumb. "Taxonomía e inspección de sistema de cómputo en las nubes". Universidad Old Dominion, 2009. Fifth International Joint Conference INC, IMS and IDC.

[6] Distributed Management Task Force (DMTF). "*Especificación del formato abierto de virtualización".* Disponible: http://www.dmtf.org/standards/published_documents/ DSP0243_1.0.0.pdf.

[7] Distributed Management Task Force (DMTF). "*Incubadora de estándares de nubes abiertas". Disponible*: http://www.dmtf.org/ about/cloud-incubator.

[8] Open Grid Forum (OGF). "Especificación de la interfaz de cómputo abierto en las nubes, Grupo de trabajo de la interfaz de cómputo abierto en las nubes". Disponible: http://www.occi-wg.org.

[9] Manifiesto de cómputo abierto en la nube. Disponible: http://www.opencloudmanifesto.org/Open%20Cloud%20Manifesto.pdf

[10] Documentación de jClouds. Disponible: http://code.google.com/p/jclouds/

[11] Documentación de Java Data Base Connectivity. Disponible: http://www.oracle.com/technetwork/java/overview-141217.html

[12] Documentación de Amazon AWS. Disponible en: http://aws.amazon.com/es/ec2/

[13] Documentación de Google AppEngine. Disponible: https://developers.google.com/appengine/

[14] Bill Claybrook. Midiendo el dilema del cómputo en las nubes estándar. Search Cloud Computing. Feb 2011. Disponible: http://searchcloudcomputing.techtarget.com/feature/Weighing-the-cloud-computing-standards-dilemma

[15] Lee Badger, Tim Grance , Robert Patt-Corner y Jeff Voas. “Cloud Computing Synopsis and Recommendations”. Publicación especial NIST 800-146. 2012

[16] Documentación del servicio de almacenamiento elástico Amazon Simple Storage Service. Disponible: http://aws.amazon.com/es/s3/

[17] Documentación del servicio de almacenamiento elástico Google Cloud Storage. Disponible: https://developers.google.com/appengine/docs/java/googlestorage/overview

[18] Documentación del framework Amazon AWS SDK. Disponible en: http://aws.amazon.com/es/sdkforjava/

[19] Documentación del framework Google AppEngine SDK. Disponible en. https://developers.google.com/appengine/docs/java/overview

[20] Jaroslav Tulach, “Diseño práctico de APIs: Confesiones de un arquitecto de frameworks Java”, Ed. Apress. EUA, 2008.

[21] “Java Code Conventions”. Sun Microsystems. 1997. Disponible en: http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

[22] Mladen A. Vouk, "Cloud Computing – Issues,Research and Implementations", Journal of Computing and Information Technology - CIT, 2008

[23] IEEE P2301 Cloud Profiles. Disponible en: http://cloudcomputing.ieee.org/standards/standards- guidance - p2301

[24] IEEE P2302 Intercloud . Disponible en: http://cloudcomputing.ieee.org/standards/standards - guidance - p2302

[25] ITU - T FG Cloud Technical Report (Parts 1 to 7). Disponible en: http://www.itu.int/en/ITU- T/jca/Cloud/Pages/default.aspx

[26] Pérez Escalera Miguel Felipe, León Chávez Miguel Ángel. “UML Model of a Standard API for Cloud Computing Application Development”. Benemérita Universidad Autónoma de Puebla, 2012. 9th International Conference on Electrical Engineering, Computing Science and automatic Control (CCE 2012).

7 APÉNDICE I. DIAGRAMAS DE PROCESOS DEL SERVICIO DE ALMACENAMIENTO PERSISTENTE EN LAS NUBES AWS Y APPENGINE

7.1 INICIALIZAR EL SERVICIO

Inicializar el servicio se refiere a la forma en que se obtiene una instancia de la clase que contiene las funciones para el acceso al servicio. En el caso de Google sólo se necesita el nombre del bucket sobre el que se va a trabajar. En Amazon es necesario un par de llaves para identificarse, esto debido a que las invocaciones se pueden realizar fuera de su nube. La forma en que se modeló el API estándar requiere un identificador del servicio y un objeto que contenga todas las propiedades para iniciar el servicio, como el nombre del bucket y las llaves.

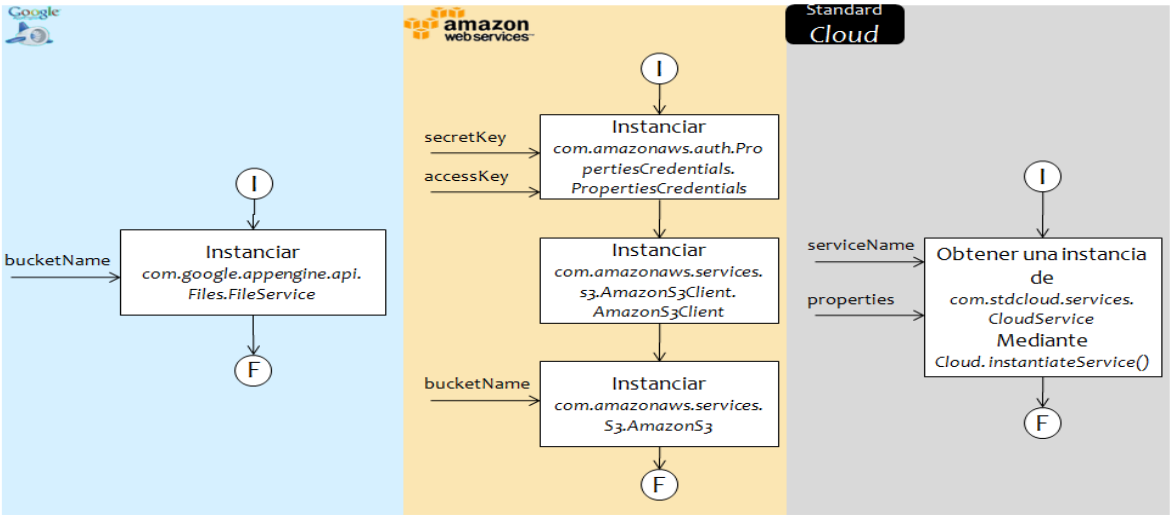


FIG. 24 COMPARATIVA DE INICIALIZACIÓN DEL SERVICIO

7.2 CREAR UN ARCHIVO EN LA NUBE

Crear un archivo es, quizá, la operación más importante. Ambas nubes permite esta operación de forma remota. En el caso de Google, requiere el nombre del bucket en el que se almacenará, una llave (el nombre del archivo) y un conjunto de parámetros auxiliares. Amazon requiere que el archivo sea creado en el sistema local y después enviado a la nube con un nombre llave y el bucket al que va. En el API estándar, tan sólo se requiere el nombre del archivo.

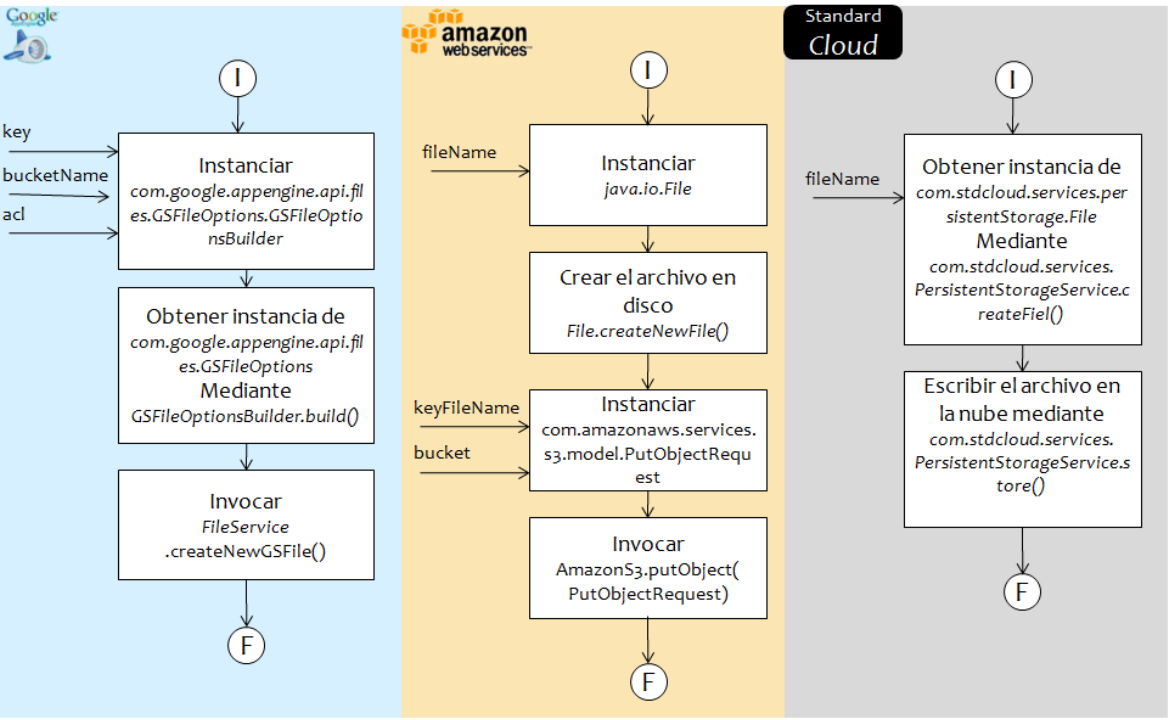


FIG. 25 COMPARATIVA DE CREACIÓN DE UN ARCHIVO

7.3 ESCRIBIR CONTENIDO A UN ARCHIVO

Escribir a un archivo es un paso posterior a la creación de este. Google requiere un nombre de archivo llave. Se hace una petición al servicio en la nube para abrir un FileChannel que permita enviar bloques de bytes de datos, finalmente hay que cerrar el canal y llamar a un método especial que previene futuras modificaciones. En Amazon se crea el archivo de forma loca con su contenido usando las primitivas de jav.io. El archivo se sube a la nube de la misma forma que en la creación. Para el API estándar sólo se crea un flujo de salida estándar de java usando el nombre del archivo. A él se le escriben los bytes de datos.

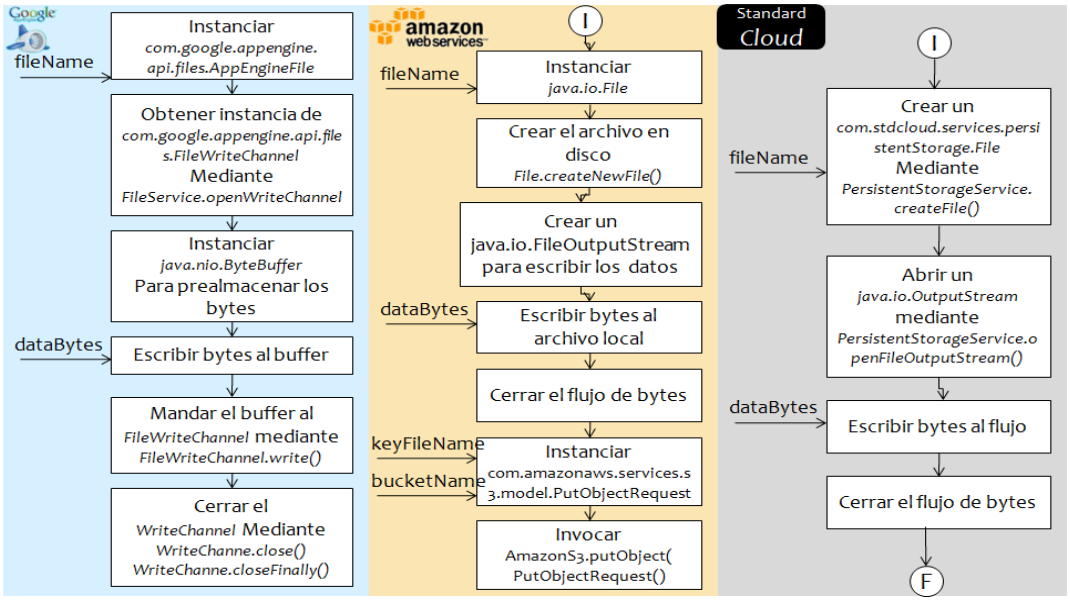


FIG. 26 COMPARATIVA DE ESCRITURA DE CONTENIDO A UN ARCHIVO

7.4 LEER CONTENIDO DE UN ARCHIVO

Ya teniendo un archivo con datos, se procede a leerlos. En Google se usa el nombre del archivo con un prefijo que determina el proveedor (/gs/) y el bucket. Con estos datos se hace una petición para abrir un *FileChannel* y se obtienen bloques de bytes de datos. Amazon requiere el nombre del archivo, el bucket y se hace una petición para abrir un flujo de entrada común para obtener sus bytes de datos. En el API estándar se usa el nombre del archivo para abrir un flujo de entrada común.

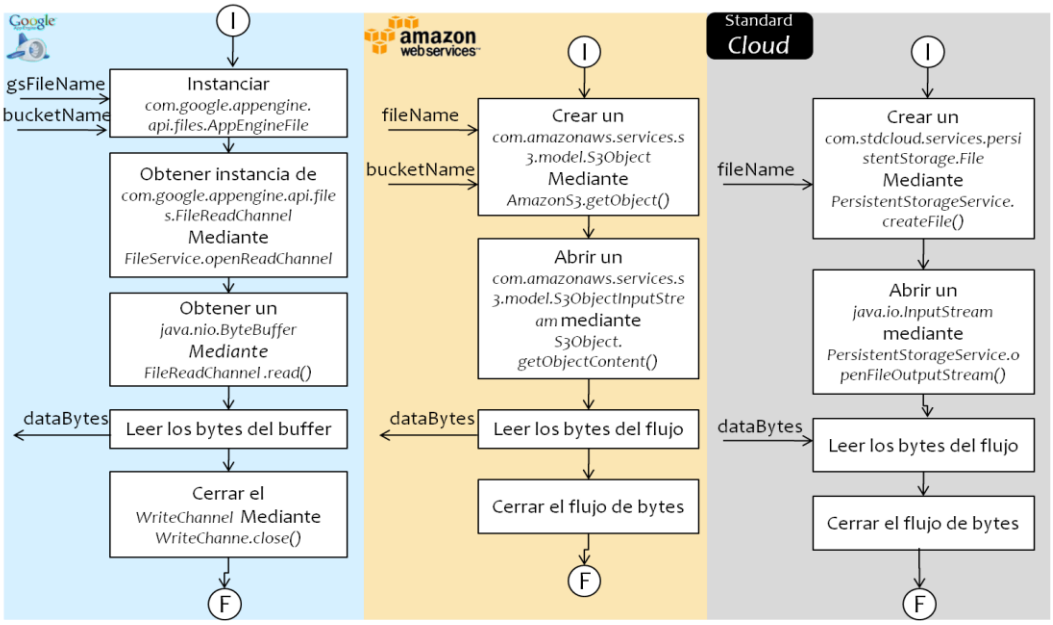


FIG. 27 COMPARATIVA DE LECTURA DE CONTENIDO A UN ARCHIVO

7.5 BORRAR UN ARCHIVO

Eliminar un archivo es una operación necesaria para no guardar datos basura. Google tiene un método complejo para la eliminación; utiliza el nombre del archivo con el prefijo de la nube, el bucket y un par de parámetros más para la invocación. Amazon simplificó este proceso con una única invocación usando el nombre del archivo. Para la nube estándar es necesario usar el nombre del archivo para solicitar su eliminación.

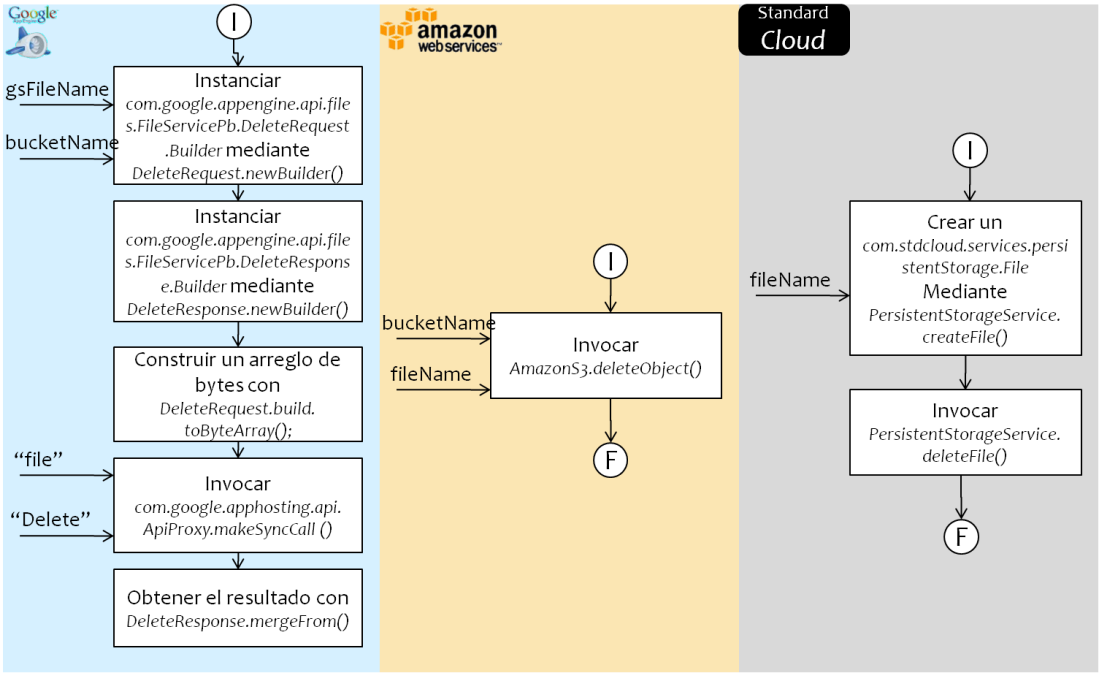


FIG. 28 COMPARATIVA DE BORRADO DE UN ARCHIVO

7.6 COPIAR/MOVER/RENOMBRAR UN ARCHIVO

Los procesos de copiar un archivo, moverlo o cambiarle el nombre se están manejando como funciones de más alto nivel que no involucran primitivas de la nube sino un proceso interno del API usando funciones que ya se presentaron antes.

El proceso para los tres casos consiste en crear el archivo de destino a partir del contenido del original. Si es un copiado de archivo, el proceso termina ahí. Si se está moviendo o renombrando, se elimina el archivo original.

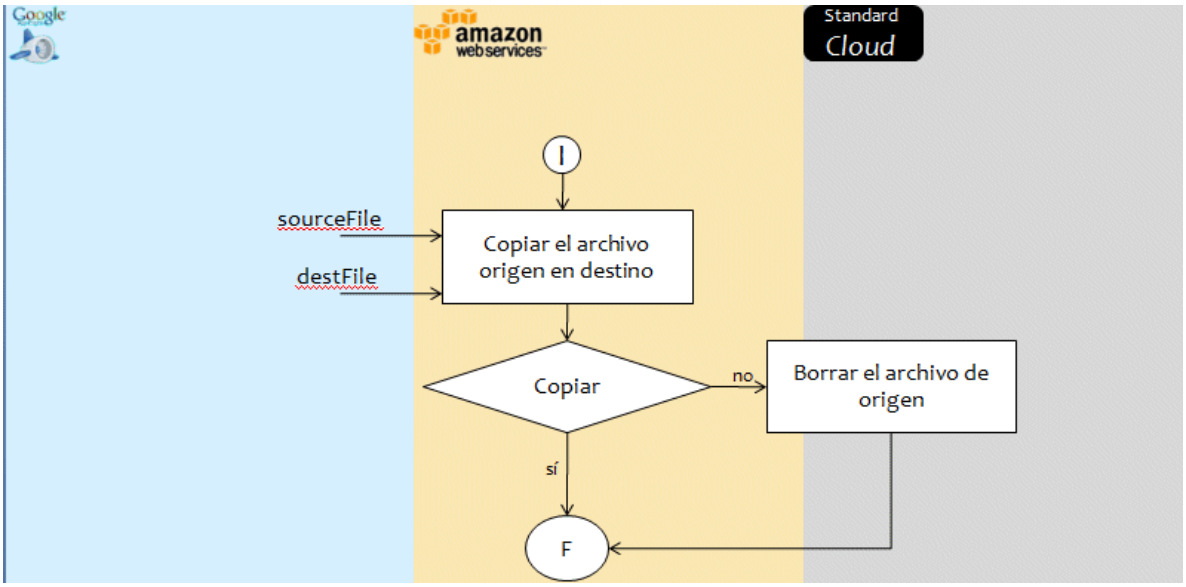
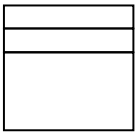
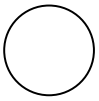
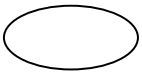
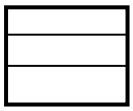
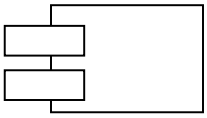
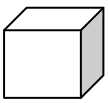


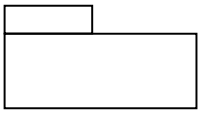
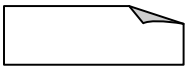
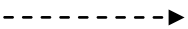
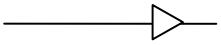
FIG. 29 COMPARATIVA DE LAS OPERACIONES ADICIONALES (COPIAR, MOVER, RENOMBRAR)

8 APÉNDICE II. BLOQUES DE CONSTRUCCIÓN UML

A continuación se muestra una tabla con los elementos que conforman los bloques de construcción del lenguaje.

TABLA 3 BLOQUES DE CONSTRUCCIÓN UML

Elemento	Descripción	Representación gráfica
Elementos estructurales		
Clase	Descripción de un conjunto de objetos que tienen los mismos atributos, operaciones, relaciones y semántica.	
Interfaz	Colección de operaciones que especifican los servicios que ofrece una clase o un componente. Puede definir todos los servicios pero carece de las implementaciones de estos.	
Colaboración	Define una interacción entre el comportamiento de una serie de elementos para proporcionar servicios mayores.	
Caso de uso	Define la secuencia de acciones que ejecuta el sistema y que provee un resultado importante para un actor del mismo.	
Clase activa	Es una clase cuyos objetos tienen uno o más procesos de ejecución concurrente y pueden necesitar actividades de control.	
Componente	Es una parte física del sistema que puede ser reemplazada por otra que cumpla las mismas características. Comúnmente necesita ser implementado. Representa una colección empaquetada de otros elementos.	
Nodo	Es un elemento físico que representa un recurso computacional que puede poseer cierta cantidad de memoria y poder de cómputo. En él pueden residir componentes que migren a otros nodos.	

Elementos de comportamiento		
Interacción	Consiste en un conjunto de mensajes que intercambia un conjunto de objetos para conseguir un resultado	
Máquina de estados	Especifica los estados por los que pasa un objeto a lo largo de su vida en respuesta a los eventos que ocurren alrededor de él.	
Elementos de agrupación		
Paquete	Es una agrupación de elementos estructurales, de comportamiento y de cualquier otro tipo. Es sólo conceptual y no existe más que en tiempo de desarrollo.	
Elementos de anotación		
Nota	Se utiliza para mostrar comentarios sobre un elemento o una colección.	
Relaciones		
Dependencia	Es una relación semántica entre dos elementos donde cambios en el elemento independiente impactan en el dependiente.	
Asociación	Es una relación estructural que define una serie de enlaces o conexiones entre objetos.	
Generalización	Es una relación de generalización o especificación donde el elemento específico (hijo) puede reemplazar los objetos del elemento general (padre).	
Realización	Es una relación semántica de clasificadores donde un clasificador define un contrato que otro clasificador se compromete a cumplir.