Universidad de las Ciencias Informáticas

Facultad 6

Titulo: Infraestructura de comunicaciones para una plataforma de Cloud Computing

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor: Alejandro Díaz Suarez

Tutores: Lic. José Albert Cruz Almaguer

Ing. Adonis Ricardo Rosales García

Junio 2010

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor del trabajo titulado:

Infraestructura de comunicación para una plataforma de Cloud Computing.

Y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente en el mes de _____ del año _____.

_____

Alejandro Díaz Suarez

_____ _____

Lic. José Albert Cruz Almaguer Ing. Adonis Ricardo Rosales García

AGRADECIMIENTOS

Agradezco a todas aquellas personas que directa e indirectamente me apoyaron en la realización del presente Trabajo de Diploma, a mis Tutores, en especial a mis padres que los quiero mucho, con todo mi corazón.

Alejandro Díaz

DEDICATORIA

Dedico este pedazo de mi a toda mi familia grandota que los quiero mucho, pero mucho, mucho. Porque estoy orgulloso de tener una familia como la nuestra.

También a mí amada Yeisa que la quiero mucho y por haber soportado todos estos meses de agonía y sufrimiento a distancia.

Alejandro Díaz

RESUMEN:

Para darle un seguimiento a la misión del centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE), la línea de Java Enterprise Edition de dicho centro ha tomado la iniciativa de desarrollar una plataforma con el modelo de Cloud Computing. Este trabajo de diploma consiste en desarrollar una infraestructura de comunicación para dicha plataforma. Después de haber hecho un profundo estudio de los distintos temas asociados, las tecnologías y métodos para el desarrollo de la misma, se definió desarrollar un Middleware Orientado a Mensajes y un API a usar por los clientes para que se puedan comunicar con la Nube, utilizando como protocolo de comunicación XMPP. Un middleware tiene todas las características para definirlo como una infraestructura, por lo que al desarrollar dicho middleware se le puede dar por cumplido el principal objetivo de este trabajo.

Palabras claves:

API, Cloud Computing, Infraestructura, Middleware.

Tabla de Contenidos

INTRODUCCIÓN:

Con el gran desarrollo de las Tecnologías de la Información y las Comunicaciones (TICs) en estos últimos años, se han creado muchos conceptos, métodos y paradigmas para aprovechar la alta capacidad de procesamiento de las computadoras del mundo actual. Los sistemas distribuidos ha sido una de las creaciones para aprovechar la alta capacidad de procesamiento.

Un sistema distribuido se define como un conjunto de computadoras que están unidas por una red de alta velocidad, estas computadoras son capaces de trabajar en la realización de una tarea común.

Son muchas las aplicaciones que se le dan a este concepto, como ejemplo tenemos a Grid Computing el cual consiste en aprovechar los recursos de los ordenadores que están conectados en una red para trabajar en un solo problema al mismo tiempo. Esto suele hacerse para resolver problemas de gran envergadura en menos tiempo. Un ejemplo de ello es SETI@Home (Búsqueda de Inteligencia Extraterrestre). En este proyecto personas de todo el mundo comparten los ciclos de reloj no utilizados de sus computadoras, para buscar señales de inteligencia.

Actualmente está en auge y perfeccionamiento el concepto de Cloud Computing (Computación en Nube, llevado al español). “Nube”, se dice que es una metáfora del término Internet. Cloud Computing es un modelo que permite acceder a las aplicaciones que residen realmente en un servidor en Internet, no en un ordenador ordinario ó en ningún otro dispositivo conectado a Internet, dejando a un lado la tarea de instalar cada aplicación que se vaya a utilizar en un ordenador. Este engloba varios conceptos como SaaS (Software as a Service), PaaS (Platform as a Service), HaaS (Hardware as a Service), entre otros.

Software as a Service (SaaS) es un modelo en que una aplicación es hosteada como un servicio a los clientes que acceden a ella a través de Internet. Los clientes solo tienen que utilizar la aplicación sin tener que instalarla. A pesar de que tienen que pagar por el servicio periódicamente, sale mucho más barato que cuando hay que pagar una vez por la instalación.

Plataforma como Servicio (PaaS) es otro modelo de entrega de aplicaciones. Proporciona todos los recursos necesarios para construir aplicaciones y servicios completamente de la Internet, sin tener que descargar ni instalar software.

Hardware como un Servicio (HaaS) simplemente ofrece el hardware, de modo que una organización que pague por el servicio, tiene la disposición de ese recurso para cualquier actividad necesaria.

En el mundo ya existen varias plataformas que implementan el modelo de Cloud Computing. Están desarrolladas fundamentalmente por empresas que tienen una gran infraestructura de recursos informáticos. Algunas de esas plataformas desarrolladas son:

- Google App Engine
- eyeOS
- Amazon EC2

Cloud Computing y la Arquitectura Orientada a Servicios (SOA) tienen una estrecha relación, puesto que las dos tienen dentro de sus objetivos brindar servicios a través de una red.

SOA es un marco estratégico de la tecnología que permite a todos los sistemas de interés que están dentro y fuera de una organización, exponer y acceder a servicios bien definidos. En esencia, SOA añade el aspecto de la agilidad en una arquitectura, ya que nos permite hacer frente a cambios en el sistema utilizando una capa de configuración en lugar de tener que volver a desarrollar constantemente estos sistemas.

Una SOA puede perfeccionarse con la ayuda de Cloud Computing en cuanto a diseño de servicios y la expansión de un servicio. Muchos de los proyectos de SOA tienden a no diseñar bien los servicios, mientras que en un modelo Cloud Computing hay que tener un buen diseño para los servicios sean rentables. La capacidad de ampliar los servicios en una SOA es normalmente un proceso costoso, en Cloud Computing los servicios están diseñados para expandirse según sea necesario.

Al mismo tiempo Cloud Computing puede adquirir una perfección con conocimientos de una SOA. Casi todas las aplicaciones están divididas en funcionalidades o módulos, respetando la función de una arquitectura SOA cada funcionalidad o módulo se puede convertir en un servicio, así al desplegar una aplicación en la plataforma Cloud Computing quedaría dividida en servicios. Esos servicios pueden estar en servidores o nodos distintos, optimizando el rendimiento ó aligerando los servidores. Como en todo proyecto se necesita una buena arquitectura para que tenga éxito, con la aplicación de una SOA, se puede alcanzar lo mejor de Cloud Computing.

El centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE) tiene como misión ofrecer los servicios de consultoría tecnológica a partir del capital intelectual que se desarrolla en la UCI, como una actividad generadora de recursos financieros para el país. Para seguir con el

cumplimiento de la misión, la línea de Java Enterprise Edition de dicho centro, ha tomado la iniciativa de desarrollar una plataforma con el modelo de Cloud Computing.

Con esta plataforma se le pudiera brindar a empresas cubanas y a otros centros de la UCI un servicio informático a un costo muy por debajo del capital que hay que destinar para su informatización, costeando solo la necesidad real.

Una plataforma de este tipo está constituida por varios servidores y cada uno de ellos puede tener varias aplicaciones prestando servicios. Para que los clientes puedan acceder a esos servicios debe existir un sistema que posibilite la comunicación entre ellos, y además debe tener el control de todas las aplicaciones desplegadas en la Nube. El sistema debe permitir una comunicación rápida y confiable para poder prestar los servicios en tiempo real y que cumpla las expectativas de los clientes.

A raíz de esto surge como **Problema Científico:** ¿Cómo lograr una comunicación confiable y rápida entre los servidores distribuidos de una plataforma de Cloud Computing?

Objeto de estudio:

Comunicación entre componentes de un sistema distribuido

Campo de acción:

Comunicación entre componentes de un sistema distribuido para el modelo Cloud Computing

Objetivo General:

Desarrollar la infraestructura de comunicaciones de una plataforma de Cloud Computing.

Objetivos Específicos:

- Diseñar una infraestructura de comunicaciones de una plataforma de Cloud Computing.
- Implementar una infraestructura de comunicaciones de una plataforma de Cloud Computing.

Tareas de investigación:

1. Estudio de las diferentes tecnologías de implementación de sistemas distribuidos.
2. Análisis de los casos más exitosos de middleware de comunicaciones.
3. Adopción de una definición de Cloud Computing adecuada a las condiciones técnicas de la UCI y de Cuba.

4. Dominio de la tecnología Erlang/OTP.
5. Análisis de los protocolos HTTP, XMPP y AMQP como alternativas para la comunicación.
6. Diseño del middleware de comunicaciones a usar entre los servidores que conformen la Nube.
7. Implementación del middleware de comunicaciones a usar entre los servidores que conformen la Nube.
8. Diseño del API de comunicaciones a usar por los clientes para comunicarse con la Nube a través del protocolo(s) identificado(s) como idóneo(s).
9. Implementación del API de comunicaciones a usar por los clientes para comunicarse con la Nube a través del protocolo(s) identificado(s) como idóneo(s).

El presente documento está estructurado en capítulos para lograr una mayor organización del contenido y mejor entendimiento, a continuación se describen los mismos.

Capítulo 1: Fundamentación teórica: Se describen los principales conceptos involucrados en la realización de la investigación, el estado del arte del tema tratado a nivel internacional así como las principales tendencias, técnicas, tecnologías, metodologías y software usados en la solución del problema.

Capítulo 2: Características del Sistema: Aquí se detallan y se explican los requisitos no funcionales, se desarrolla la concepción del concepto Cloud Computing con las condiciones tecnológicas de la UCI y Cuba. También se toma el tema de la arquitectura del sistema relacionada con los comportamientos OTP que presenta la tecnología Erlang.

Capítulo 3: Exploración y Planificación del sistema: Se detallan los artefactos relacionados con la exploración y planificación de la aplicación.

Capítulo 4: Diseño, Implementación y prueba: Se describen los artefactos relacionados con el diseño, la implementación y pruebas de la aplicación.

FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se analizarán los conceptos y temas relacionados de Cloud Computing, Sistemas distribuidos, Middleware y Arquitectura Orientada a Servicios (SOA). Además también se van a analizar las herramientas de desarrollo, protocolos de comunicación, metodologías de desarrollo de software y las herramientas CASE, seleccionando las tecnologías más factibles para desarrollar la infraestructura de comunicación de una plataforma Cloud Computing.

1.2 Cloud Computing

El tema de discusión de Cloud Computing se puede encontrar en cualquier parte. Casi todos hablan y discuten sobre el tema, pero no todos tienen la misma opinión, en el caso de que tomáramos al azar a varios profesionales y preguntáramos," ¿Qué es Cloud Computing?", seguramente cada uno daría una respuesta diferente.

A continuación se dará una breve explicación sobre que es Cloud Computing, en qué consiste, cómo trabaja, los conceptos que se incluyen en este tema, entre otras cosas de interés.

1.2.1 ¿Cloud Computing?

Cloud Computing recibe su nombre como una metáfora de la Internet. Normalmente, la Internet se representa en diagramas de red como una Nube. El icono de la Nube representa el "etc." para el resto de la solución del diagrama que se muestra en la figura siguiente.

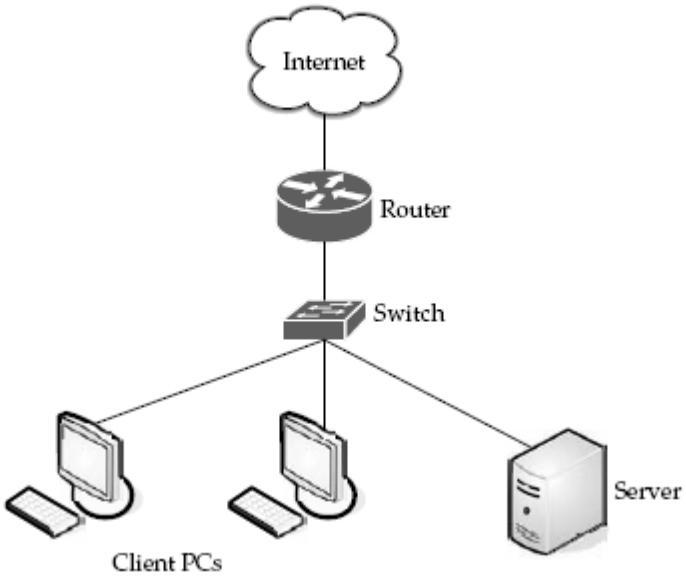


Figura 1. Representación grafica de Cloud Computing

En esencia, Cloud Computing es un concepto que permite acceder a las aplicaciones que residen realmente en un servidor en Internet, no en un ordenador ordinario ó en un dispositivo. Esto trae consigo muchos beneficios: no se tiene la preocupación de tener un disco de instalación de las aplicaciones, ni comprar las licencias de dichas aplicaciones e incluso actualizarlas una vez ya instaladas, solo tiene que consumir el servicio de esas aplicaciones a través de Internet. El beneficio de este concepto es que otra empresa es la que costea los servidores, el almacenamiento, las actualizaciones de las aplicaciones, la electricidad, sistema de enfriamiento, entre otras cosas, mientras que solo se paga por el servicio, disminuyendo los costos de inversión. También los usuarios que tienen acceso al servicio, pueden hacerlo a través de cualquier punto de red que tenga conexión con la empresa que brinde dicho servicio.

1.2.2 Debilidades

Cloud Computing no está exenta de los problemas. En la figura siguiente se muestra dos vías de fallo, una por la parte del cliente y la otra por la parte de donde se brinda el servicio.

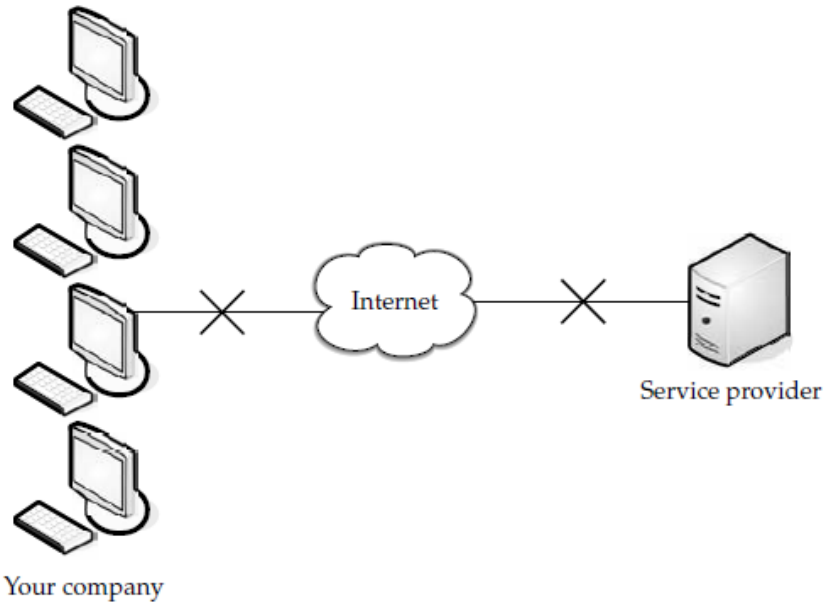


Figura 2. Vías de fallos de Cloud Computing

Por la parte del cliente, puede que se caiga la conexión a Internet por cualquier motivo, ya sea por falta de electricidad, problemas con el cable de red, mala configuración de la red, entre otras muchas causas por la cual se puede caer Internet en una empresa.

En cuanto a la otra parte, puede ser que suceda lo mismo. Pero además, puede que las aplicaciones estén fallando y haya que repararlas. Esto descontenta al cliente, puesto que reparas dichas aplicaciones puede tardar varias horas.

1.2.3 Componentes

En el sentido topológico, la solución de la Cloud Computing está dada por tres elementos: los clientes, el centro de datos, y los servidores distribuidos.

Los clientes generalmente son aquellos equipos que se encuentran en un escritorio. Pero también podrían ser laptops, teléfonos móviles o PDA, en fin cualquier dispositivo con que los usuarios finales interactúan con la gestión de su información en la Nube.

El centro de datos es la colección de servidores donde se encuentran las aplicaciones subscritas. Una tendencia creciente en el mundo de las Tecnologías de la Información es la vitalización de los servidores, es decir, el software puede ser instalado permitiendo múltiples instancias de servidores virtuales. De esta manera, se puede tener la cantidad necesaria de servidores virtuales en un servidor

físico. Al tener varios servidores permite que si deja de funcionar uno los otros pueden asumir la responsabilidad, encargándose de las funciones que realizaba el servidor, mientras este se repara.

Los servidores no tienen que estar presentes en un mismo lugar en una arquitectura de Cloud Computing. A menudo, los servidores están en lugares geográficamente diferentes. Pero, en la Nube, estos servidores trabajan como si estuvieran uno al lado del otro. Esto le da al proveedor de servicios una mayor flexibilidad en las opciones y en la seguridad. Por ejemplo, Amazon tiene su solución de Nube en servidores por todo el mundo, si pasa algo en un sitio determinado, causando un fallo, se puede acceder al servicio a través de otro sitio.

Hay muchas vías de desplegar una infraestructura. Depende de la aplicación y de cómo el proveedor construya la solución de la Nube, esta se ajusta en dependencia de las necesidades, y es una de las principales ventajas para el uso de la Nube.

1.2.4 La Grid y Cloud Computing

Computación Grid es frecuentemente confundida con Cloud Computing, pero son muy diferentes. Computación Grid aplica los recursos de muchos ordenadores en una red para trabajar en un solo problema al mismo tiempo. Esto suele hacerse para abordar un problema científico o técnico.

Computación Grid es atractiva por varias razones:

- Es una manera rentable de utilizar una determinada cantidad de recursos informáticos.
- Es una manera de resolver problemas que necesitan una enorme cantidad de potencia de cálculo.
- Los recursos de varios ordenadores pueden ser compartidos en forma cooperativa, sin que un equipo gestione recursos del otro.

Entonces, ¿Computación Grid y Cloud Computing tiene que ver una con otra? No mucho directamente, ya que funcionan de manera fundamentalmente diferente. Además, Cloud Computing es justo lo contrario. Permite que diversas aplicaciones pequeñas se ejecuten al mismo tiempo.

1.2.5 Servicios

El termino *servicios* en Cloud Computing es el concepto de ser capaz de utilizar componentes reutilizables en toda la red de un proveedor. Esto es conocido como "*as a Service*". Las ofertas que incluyen "*as a Service*" como un sufijo, incluyen las características siguientes:

- Bajas barreras de entrada, poniéndolas a disposición de pequeñas empresas.
- Gran escalabilidad.
- Multiusuario, que permite que los recursos sean compartidos por muchos usuarios.
- Independencia del dispositivo, que permite a los usuarios acceder a los sistemas en diferentes hardwares

SaaS

Software as a Service (SaaS) es un modelo en que una aplicación es hosteada como un servicio a los clientes que acceden a ella a través de Internet. Los proveedores son los encargados de actualizar la aplicación, la integración con otros sistemas y del mantenimiento de la infraestructura de funcionamiento. Los clientes solo tienen que utilizar la aplicación sin tener que instalarla y a pesar de que tienen que pagar por el servicio periódicamente, sale mucho más barato que cuando hay que pagar una vez por la instalación.

Para los vendedores, SaaS tiene el atractivo de ofrecer una mayor protección de su propiedad intelectual, así como la creación de un flujo continuo de ingresos.

Hay muchos tipos de software que se prestan para el modelo de SaaS. Por lo general, el software que realiza una tarea sencilla sin mucha necesidad de interactuar con otros sistemas los hace candidatos ideales para SaaS. Los clientes que no están dispuestos a llevar a cabo el desarrollo de software, pero tienen necesidad de aplicaciones de alta potencia también pueden beneficiarse de SaaS. Algunas de estas aplicaciones incluyen:

- Gestión de recursos de clientes (CRM)
- Videoconferencia
- Gestión de Servicios TI
- Contabilidad
- Analítica Web
- Gestión de contenidos web

PaaS

Plataforma como Servicio (PaaS) es otro modelo de entrega de aplicaciones. Proporciona todos los recursos necesarios para construir aplicaciones y servicios completamente de la Internet, sin tener que descargar ni instalar software.

Los servicios de PaaS incluyen diseño de aplicaciones, desarrollo, pruebas, despliegue y hosting. Hay otros servicios que incluyen la colaboración en equipo, la integración de servicios web, integración de bases de datos, seguridad, escalabilidad, el almacenamiento, la gestión del Estado, y las versiones.

Una de las fallas de PaaS pudiera ser la falta de interoperabilidad y la portabilidad entre los proveedores. Es decir, si se crea una aplicación con un proveedor de Nubes y después se decide trasladarse a otro proveedor, puede que no se logre hacerlo o si tendrá que pagar un alto precio.

HaaS

Hardware como un Servicio (HaaS) es el siguiente formulario de servicios en Cloud Computing. En el caso de SaaS y PaaS están ofreciendo aplicaciones para los clientes. HaaS Simplemente ofrece el hardware de modo que una organización que pague por el servicio puede poner ó hacer lo que quiera con él.

1.3 Sistemas Distribuidos

Como habíamos aclarado al inicio, un sistema distribuido se define como un conjunto de computadoras que están unidas por una red de alta velocidad, estas computadoras son capaces de trabajar en la realización de una tarea común, todas a la misma vez. [1]

Los sistemas distribuidos se pueden implementar o desarrollar en una red de área local con pocos nodos independientes. También admiten la intrusión de nuevos nodos, así como el crecimiento de toda la red, debido a su escalabilidad. Un ejemplo de ello es Internet, que comenzó por varios nodos dentro de los Estados Unidos y actualmente la red abarca al mundo entero.

Las aplicaciones distribuidas deben estar caracterizadas por la fiabilidad, la seguridad y privacidad en el sistema, debido a que son usadas por usuarios comunes en centros de investigación, universidades y empresas. Deben permitir el acceso a una información por varios usuarios al mismo tiempo, que el tiempo de respuesta sea la más rápida posible y deben permitir la integración con otros sistemas y el crecimiento de la misma.

1.3.1 Características de Sistemas Distribuidos

Concurrencia:

La concurrencia permite la utilización simultánea de los recursos disponibles por los usuarios conectados a la red. [1]

En los sistemas distribuidos se pueden levantar la misma cantidad de procesos que la cantidad de maquinas conectadas en la red. Estos procesos se ejecutan en paralelo, no tienen que esperar a que termine de ejecutarse un proceso para comenzar el otro. Inician su función cuando se ejecuten, ya que son independientes cada uno de ellos.

En un sistema distribuido basado en el modelo de compartición de recursos, la posibilidad de ejecución paralela ocurre por dos razones:

1. Muchos usuarios interactúan simultáneamente con programas de aplicación.
2. Muchos procesos servidores se ejecutan concurrentemente, cada uno respondiendo a diferentes peticiones de los procesos clientes.

El primero de los casos es menos conflictivo porque las aplicaciones son casi siempre ejecutadas aisladamente desde el entorno de trabajo del usuario y no se dificulta su función al ejecutarse otras aplicaciones por otros usuarios desde sus ubicaciones de trabajo.

El otro caso surge por haber en la red uno o varios procesos servidores para cada tipo de recurso. Estos procesos se ejecutan en distintas máquinas, de manera que se están ejecutando en paralelo diversos servidores, junto con diversos programas de aplicación. Las peticiones realizadas por los clientes a los recursos de un servidor pueden ser puestas en espera para ser procesadas paulatinamente o pueden ser procesadas varias al mismo tiempo. Cuando esto ocurre los procesos servidores deben sincronizar sus acciones para asegurarse de que no existen conflictos. La sincronización debe ser cuidadosamente planeada para asegurar que no se pierden los beneficios de la concurrencia. [1]

Carencia de reloj global:

No se utiliza un temporizador general en las coordinaciones para el envío de mensajes. [1]

Compartición de recursos:

En un sistema distribuido todo se conecta funcionando como un solo individuo, por eso es posible la compartición de componentes hardware como discos e impresoras hasta elementos software como ficheros, ventanas, bases de datos y otros objetos.

En un sistema distribuido las computadoras acceden a los recursos que están físicamente encapsulados en solo una computadora, y son las únicas que lo pueden hacer en toda la red.

Para que sea efectiva la compartición de recursos, debe ser manejada por un programa que ofrezca una interfaz de comunicación permitiendo el acceso, manipulación y actualización de una manera fiable y consistente, esto es lo que recibe el nombre de gestor de recursos que no son más que un módulo software que maneja un conjunto de elementos de un tipo en particular [1]. Todos estos tipos requieren de políticas y métodos específicos junto con requisitos específicos, los cuales incluyen la provisión de un esquema de nombres para cada clase de recurso.

La perspectiva de que un sistema distribuido se puede ver como un conjunto de gestores de recursos y programas que utilizan esos recursos y a su vez los usuarios utilizan los gestores para acceder a los recursos compartidos del sistema nos lleva a dos modelos de sistemas distribuidos:

1. Modelo Cliente-Servidor.
2. Modelo basado en objetos.

Apertura (Openness):

La apertura de los sistemas distribuidos se determina por el grado hacia el que nuevos servicios de compartición de recursos se pueden añadir sin perjudicar ni duplicar a los ya existentes. [1]

Escalabilidad:

Los sistemas distribuidos tienen una gran escalabilidad. Porque cuando el tamaño y la complejidad de las redes de ordenadores crece, el sistema distribuido seguirá siendo eficiente y útil con nuevas configuraciones de la red. [1]

Un sistema distribuido soporta desde la escala más pequeña, que consta de dos ordenadores y un servidor de ficheros, hasta la mayor escala posible. A menudo se conectan a una red existente otras redes de igual o mayor magnitud, formando lo que se denomina *Internetworks*.

La escalabilidad de los sistemas distribuidos está íntimamente ligada a sus diseños y no solo es un problema a la hora de prestaciones de red o de hardware. El diseño de un sistema distribuido debe entender la necesidad de la escalabilidad, sino existirán limitaciones muy serias y problemas que traen consigo un diseño no escalable.

La necesidad de que los sistemas distribuidos tengan una gran escalabilidad ha conducido a una filosofía de diseño en que cualquier recurso puede extenderse para dar servicio a todos los usuarios. Un ejemplo es cuando aumenta la frecuencia de acceso a los ficheros debido al aumento de usuarios y estaciones de trabajo en un sistema distribuido y se debe añadir ordenadores servidores para evitar el cuello de botella que se produciría si todas las peticiones tuvieran que ser manejadas por un servidor de fichero. Por lo que el sistema tendría que estar diseñado para trabajar con replicas de ficheros en varios servidores.

En resumidas cuentas, el trabajo necesario para manipular una petición de un recurso compartido debe ser independiente de la cantidad de usuarios y de la magnitud de la red. Las técnicas necesarias para conseguir estos objetivos incluyen el uso de datos replicados, la técnica asociada de caching, y el uso de múltiples servidores para manejar ciertas tareas, aprovechando la concurrencia para permitir una mayor productividad.

Tolerancia a fallos:

Cuando ocurre un fallo de cualquier componente de un sistema distribuido, no afecta a los demás componentes conectados a la red. Cada uno de ellos es independiente del otro.

Los sistemas informáticos no siempre realizan sus actividades satisfactoriamente, a veces presentan fallos. Pueden dejar de funcionar por falta de corriente, presentar errores internamente, mostrar resultados erróneos, en fin, existen varias razones por las cual pudieran presentar fallos. Existen dos cuestiones, complementarias entre sí, las cuales los sistemas tolerantes a fallos se basan en ellas: *Redundancia hardware* (consiste en usar componentes redundantes) y *Recuperación del software* (Los diseños de programas tienen que ser capaces de recuperarse de los fallos).

Los sistemas distribuidos también presentan un alto grado de disponibilidad en la vertiente de fallos hardware. Si se produce un fallo simple en una computadora, no tiene relevancia porque el usuario se puede cambiar de lugar. Si falla uno de los componentes de un sistema distribuido, solo afecta el trabajo que se estaba haciendo, y un proceso servidor se puede ejecutar en otra máquina de la red.

Transparencia:

La transparencia se define como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes de un sistema distribuido, de manera que el sistema se percibe como un todo, en vez de una colección de componentes independientes. La transparencia ejerce una gran influencia en el diseño del software de sistema. [1]

Existen ocho formas de transparencia:

1. Transparencia de Acceso: Se accede a los objetos de información remotos de la misma manera que se accede a los objetos de información locales.
2. Transparencia de Localización: Se accede a los objetos sin conocimiento de su ubicación.
3. Transparencia de Concurrencia: Se ejecutan procesos concurrentemente utilizando objetos de información compartidos, sin haber interferencia entre ellos.
4. Transparencia de Replicación: Se utiliza múltiples instancias de los objetos de información.
5. Transparencia de Fallos: Si falla un componente en el sistema distribuido, los usuarios y aplicaciones siguen haciendo su trabajo sin afectaciones.
6. Transparencia de Migración: Se pueden trasladar los objetos de información dentro del sistema sin afectar el trabajo de los usuarios y de otras aplicaciones.
7. Transparencia de Prestaciones: Se puede reconfigurar el sistema para mejoras las prestaciones.
8. Transparencia de Escalado: Se puede ampliar el sistema y las aplicaciones sin afectar la estructura del sistema o los algoritmos de las aplicaciones.

Las transparencias de Acceso y Localización a menudo se les denominan transparencias de red. Son las más importantes de las ocho que existen, con su presencia o ausencia afectan a la utilización de los recursos distribuidos.

Otras características:

- Cada elemento de cómputo tiene su propia memoria y su propio Sistema Operativo.
- Control de recursos locales y remotos.
- Sistemas Abiertos (Facilidades de cambio y crecimiento).
- Plataforma no estándar (Unix, NT, Intel, RISC, Etc.).
- Medios de comunicación (Redes, Protocolos, Dispositivos, Etc.
- Capacidad de Procesamiento en paralelo.

- Dispersión y parcialidad.

1.3.2 Ventajas y Desventajas

Ventajas: [2]

Con respecto a Sistemas Centralizados:

- Una de las ventajas de estos sistemas es la economía porque los microprocesadores ofrecen mejor proporción precio/rendimiento que los mainframes.
- El trabajo en conjunto.
- Mayor confiabilidad porque si una máquina se descompone, el sistema puede sobrevivir como un todo.
- Capacidad de crecimiento incremental. Se puede incrementar la potencia del sistema adicionando procesadores al sistema según las necesidades.

Con respecto a PCs independientes:

- Se pueden compartir recursos.
- El trabajo es más rápido porque un sistema distribuido puede tener mayor poder de cómputo que un mainframes.

Desventajas: [2]

- Existen una gran diversidad de criterios sobre el diseño, implementación y uso del software distribuido.
- Existen problemas en la red de comunicación debido a la pérdida de mensajes y la saturación en el tráfico.
- Pueden surgir problemas en la seguridad de la máquina al compartir los recursos.

1.4 Middleware

En un sistema distribuido existen muchos componentes, los cuales tienen que comunicarse para poder realizar las tareas y brindar los servicios para los cuales fue diseñado el sistema. Por esto, los sistemas distribuidos contienen un subsistema de comunicación que se encarga de gestionar los

mensajes y trasladar información de un componente a otro. A causa de esto es que se le dedica un espacio en este capítulo a ese subsistema que se le denomina *Middleware*.

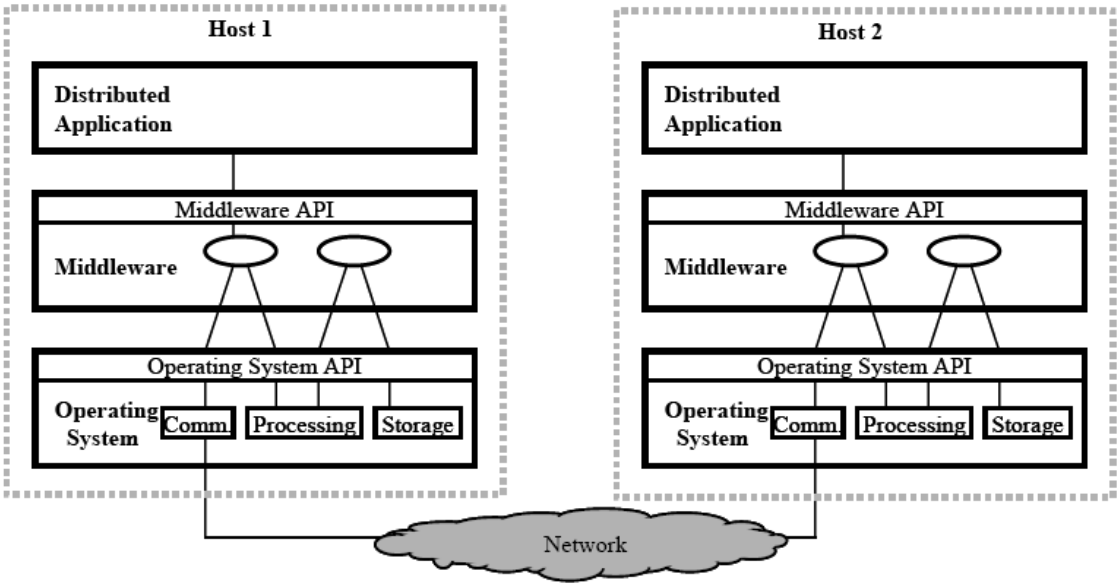


Figura 3. La Capa del Middleware ubicada en contexto.

1.4.1 Definición

A continuación se darán varias definiciones de middleware por varios criterios.

Podemos decir que middleware es el conjunto de servicios que permiten a las aplicaciones distribuidas interoperar en redes LAN o WAN. Enmascara la complejidad del sistema tanto para los usuarios finales como para los desarrolladores de las aplicaciones, proporcionando el acceso transparente a los servicios que se encuentran a través de los recursos del sistema (computadoras, impresoras, módems, software, etc.).[3]

El Middleware es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El Middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipo de servicios de middleware. [3]

Middleware es un término general para cualquier programación que sirve para unir o trabajar como puente entre dos programas separados, y que por lo general ya existen. La mensajería es un servicio común provisto por programas de middleware de tal manera que diferentes aplicaciones se puedan comunicar. (Daccach, José Camilo)[3]

Middleware se refiere a una capa de software entre los servicios de la red y las aplicaciones, encargadas de proporcionar servicios como identificación, autenticación, autorización, directorios y movilidad. (López, Eric)[3]

1.4.2 Origen

El término middleware se utilizó por primera vez en 1968. Está registrado en el Reporte de la Conferencia de Ingeniería de Software NATO, donde se denominó middleware al software mediador entre las aplicaciones y el sistema operativo.

En la década del 80 fue donde alcanzo una gran popularidad porque constituía la solución de cómo integrar las nuevas aplicaciones con los sistemas heredados. Gracias a esto el término middleware evoluciona y hace también que evolucione un conjunto de paradigmas y servicios, facilitando el manejo de los mensajes y la integración entre las aplicaciones.

Podemos ver la evolución a partir de la década del 80 dividida en tres categorías de productos:

Estaciones de mensajería (EM):

Las EM le quita la carga a las aplicaciones de mantener los parámetros de conexión y de manipular la mensajería directamente. Lo único que tienen que hacer las aplicaciones es saber cómo intercambiar mensajes con una EM, y ella se encarga de hacer llegar los mensajes a su destino de forma rápida y segura.

Las EM normalmente ocupan el centro de una topología en estrella. Es ahí donde radica su desventaja, ya que al fallar la EM todos los componentes del sistema se quedarían incomunicados. Es por eso que deben de tomar todas las medidas para que no fallen las EM, y es recomendable ubicarlas en equipos de alto rendimiento y de gran disponibilidad.

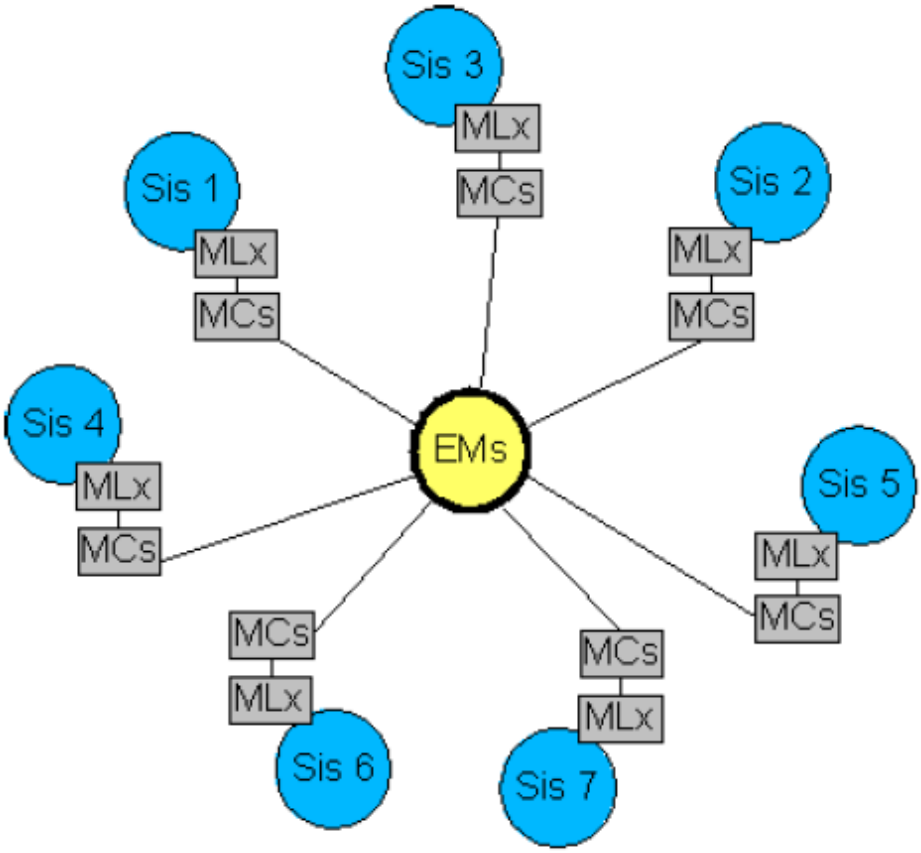


Figura 4. Topología de red de una Estación de Mensajería

Las EM aparecieron a finales de los 80 y cada proveedor las desarrollaba con su diseño, no había un estándar para el diseño de estas. Esto provocó que la interacción entre varias EM de distintos fabricantes fuera nula o limitada, porque cada una tenía su propia forma de gestionar la comunicación.

Motores de Integración (Mint)

Los MInt aparecieron a finales de los años 90 como una evolución de las EM. Los MInt heredan todas las características de las EM a las que agregan un nuevo tipo de funcionalidades, basadas en la interpretación y tratamiento de los mensajes. Entre los nuevos servicios se pueden destacar la traducción de mensajes y el enrutamiento basado en el contenido.

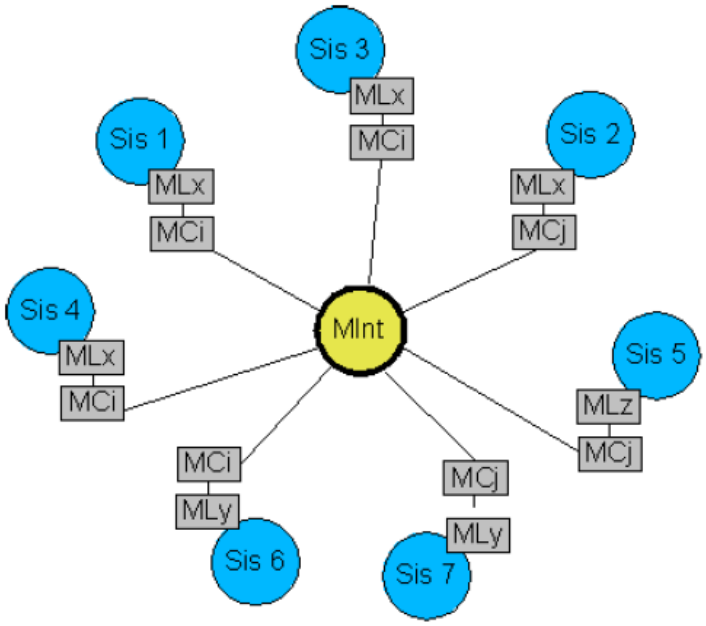


Figura 5. Topología de Red de un Motor de Integración.

La traducción de mensajes es muy importante en este entorno, debido a que es prácticamente imposible que los sistemas utilicen el mismo lenguaje. En cuanto mayor sea la integración en un proyecto, menor es el control sobre los sistemas que interactúan en él y la probabilidad de que el escenario sea multilenguaje es mayor. Los Mint están destinados para ordenar todo esto haciendo que las barreras lingüísticas no sean un obstáculo para las interacciones.

Como se ha señalado anteriormente en el núcleo de todo MInt se encuentra una EM esto hace que además de heredar sus posibilidades también herede sus limitaciones y una de ellas es que su dominio de aplicación natural está limitado a una red de área local (LAN). [3]

Buses de interacción (ESB):

A partir del año 2002 es que se viene manejando este concepto. Este es un tipo de Middleware diseñado para superar las limitaciones de los MInt y responder a las necesidades de integración de las empresas a una escala superior a la considerada hasta ese momento [3].

Cuando se habla de ESB existen dos conceptos importantes que no se pueden excluir, bus y servicios. Los ESB pueden estar físicamente distribuidos pero constituyen una única unidad lógica y todos los sistemas conectados comparten el mismo espacio de direccionamiento. Los servicios de integración están disponibles para cualquier sistema conectado al bus, no están físicamente ligados a ninguna

ubicación y pueden replicarse en aras de la eficiencia y de la seguridad. Entre estos servicios están los de validación, transformación y traducción de mensajes, enrutamiento basado en el contenido, definición y control de procesos de negocio, autenticación, autorización y auditoría, monitorización, administración, y otros.

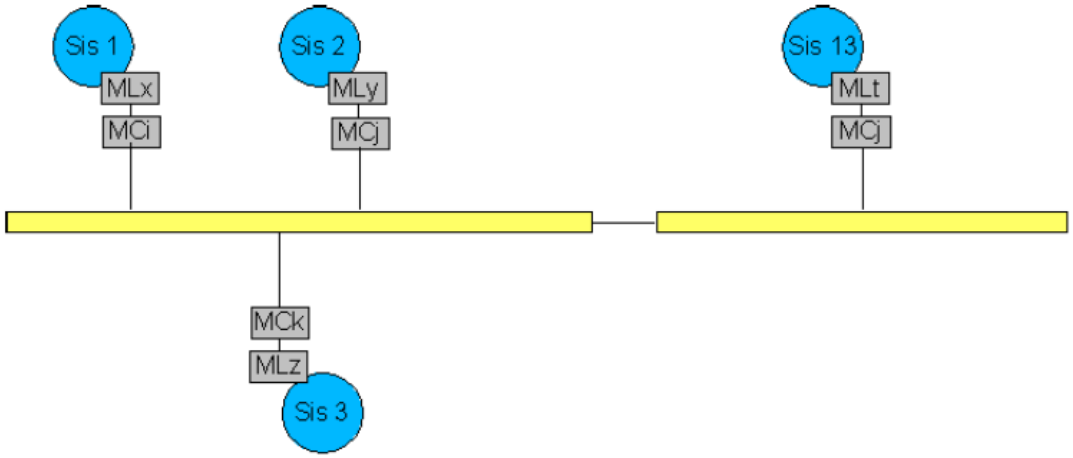


Figura 6. Topología de Red de los Buses de Interacción.

En cuanto a los servicios todos los sistemas conectados al ESB son proveedores o consumidores de servicios dónde los ESB ofrecen el recubrimiento adecuado en caso de que algún sistema sea ajeno a este concepto. De esta forma los ESB hacen posible el despliegue de Arquitecturas Orientadas a Servicios (SOA) sin exigir que los sistemas participantes ofrezcan una interfaz basada en servicios.

La columna vertebral de todo ESB es un MOM aunque dicha tecnología siga siendo propietaria pero es muy robusta y está muy probada. Lo único que debe saber cada sistema participante es cómo enviar y recibir mensajes al ESB. A diferencia de lo que sucedía con las EM, los procedimientos se basan en estándares y pueden ser muy variados, desde un acceso directo a una base de datos al uso de servicios web, pasando por protocolos propietarios o incluso el intercambio de ficheros. En cualquier caso el ESB es el que hace el esfuerzo de adaptarse a lo que el sistema en cuestión sea capaz de manejar. Para este propósito los ESB disponen de una amplia variedad de adaptadores: JDBC, FTP, SMTP, HTTP, RMI/IIOP, acceso a ficheros planos etc.

En un ESB la caída de un segmento no compromete el funcionamiento de los demás. Los ESB pueden configurarse de manera que existan servicios, rutas y segmentos alternativos, lo que junto a la ausencia de puntos de control y administración únicos hace que sean infraestructuras capaces de funcionar en régimen de alta disponibilidad.

Todos los servicios de integración que aportan los MInt se pueden encontrar en los ESB, pero más y mejor resueltos, en particular los que tienen que ver con los procesos de negocio. Algunos

proveedores incluyen también una capa que permite diseñar interfaces de usuario, con lo que abren la puerta a un nuevo producto de integración que son las aplicaciones compuestas que se construyen combinando los servicios disponibles a lo largo del bus.

Debido a la cantidad de especificaciones utilizadas por los ESB y la competencia que existe entre las diferentes normas, ya sean avaladas por alguna organización u otras que se encuentran en su primera versión, se dificulta la interoperabilidad entre buses de distintos proveedores y la posibilidad de adquirir conectores proporcionados por terceros. En los próximos años cabe esperar que el panorama mejore y que esto redunde en un abaratamiento de los productos y en una mayor independencia de los proveedores. [3]

1.4.3 Tipos de middleware

Existen varias clasificaciones de middleware que están en dependencia de su escalabilidad y la tolerancia a fallos. A continuación se mostrará varios tipos de middleware:

1. Distributed Tuples (DT): Middleware para acceso a base de datos que permite desarrollar sistemas independientes del manejador de base de datos que lo soporta.
2. Remote Procedure Call (RPC): Es el Middleware diseñado como servicio síncrono para permitir la gestión remota de redes. Esconde las operaciones de envío y recepción bajo el aspecto de una llamada convencional a una rutina o procedimiento. Los RPC tienen la misma semántica que las llamadas a procedimientos ordinarios; es decir, se realiza la llamada y se pasa el control al procedimiento servidor; cuando éste devuelve el resultado, el cliente recupera el control. El software que soporta RPC debe ocuparse de tres tareas importantes: la interfaz del servicio, la búsqueda del servidor, y la gestión de comunicación.
3. Messaging Oriented Middleware (MOM): Middleware orientado a mensajes; está diseñado para el servicio de mensajes con tecnología asíncrona. Permite el envío de mensajes entre aplicaciones, las aplicaciones sólo ponen y sacan mensajes de las colas, no se conectan. El cliente y el servidor pueden ejecutarse en diferentes tiempos (mensajes asíncronos), por lo que no necesariamente se requiere respuesta.
4. Distributed Object Middleware (DOM): Middleware para tecnologías orientadas a objetos; los objetos piden servicio a otros objetos que se encuentran en la red. Se encarga de establecer comunicación entre los clientes y los objetos de forma transparente respecto a la distribución. Permite localizar a un objeto remoto dada una referencia a ese objeto. El núcleo de estos Middleware es el Object Request Broker (ORB).

4.1 Common Object Request Broker Arquitecture (CORBA): Es un modelo de soporte para la programación distribuida orientada a objetos. Hace posible que los objetos interactúen a través de lenguajes de programación, protocolos de comunicación y plataformas heterogéneas. Este modelo no especifica cómo hacer el soporte, sino qué debe hacer, basado en cinco aspectos de los sistemas distribuidos :

- Interface Definition Lenguage (IDL): Permite la descripción de la interfaz que ofrece un objeto.
- Corba Services (Servicios CORBA): Complementan a los objetos que sirven para la construcción de aplicaciones.
- Corba Facilities (Facilidades CORBA): Cubren servicios de alto nivel, como interfaces, administración de sistemas y redes.
- Corba Domains (Interfaces de dominio CORBA): Proveen funcionalidad a usuarios finales en áreas de interés particular.
- General Inter-ORB Protocol (GIOP). Define los mensajes y el empaquetado de datos que se transmiten entre objetos. Además, define su implementación sobre otros protocolos.

4.2 Remote Method Invocation (RMI): Posee la misma finalidad que el RPC: invocar de la manera más transparente posible un servicio en una máquina virtual distinta a la que reside el cliente (en la misma máquina física pueden existir varias máquinas virtuales de Java). La diferencia entre estas dos tecnologías radica en que RPC se utiliza en diseños no orientados a objetos, mientras que RMI está soportado por el lenguaje orientado a objetos Java. RMI es un Middleware específico que permite a clientes invocar métodos de objetos como si estuviesen en la misma máquina virtual.

4.3 Distributed Component Object Model (DCOM): Al igual que CORBA y RMI, permiten la comunicación de objetos, pero únicamente para las diferentes versiones del sistema operativo Windows. DCOM surge como evolución de Object Linking and Embedding (OLE) y Component Object Model (COM).

5. Transaction Processing Monitor (TP Monitor): Middleware para Procesamiento de Transacciones ya que facilita la conectividad y el acceso a un gran número de usuarios con servicios de back-end limitados. Este tipo de Middleware requiere del soporte de un monitor; es decir, un programa que supervise las transacciones entre procesos, con el propósito de asegurar el éxito de la transacción, o en caso de ocurrir un error, tomar acciones apropiadas. Su principal uso es coordinar el flujo de solicitudes entre los dispositivos y las aplicaciones que procesan esas solicitudes.

6. Database Access Technology (DBAT): Son las Application Programming Interface (API) creando una capa transparente para el acceso a base de datos, ocultando la complejidad dada por el manejador de base de datos.

6.1 Java Database Conectivity (JDBC): Es una API que facilita programar el acceso a bases de datos para Java sin que se tenga en cuenta a que Servidor se dirige

6.2 Open Database Conectivity (ODBC): Es una API abierta para acceder a bases de datos. Especifica un conjunto de funciones para manejar conexiones a bases de datos, ejecutar declaraciones SQL y consultar las capacidades del sistema de base de datos.

7. Componente Oriented Framework (COF): Este Middleware está soportado en el modelo de desarrollo de aplicaciones basado en componentes, permite la creación de una aplicación como conjunto de componentes reusables. Los componentes son una evolución del software orientado a objetos para dar respuesta a la reusabilidad. Los Middleware basados en componentes permiten a éstos "pegarse" con otros componentes para lograr la integración.

8. Directory Services (DS): Se basa en directorios que permiten reducir costos administrativos, simplifican y distribuyen tareas administrativas, reducen el número de contraseñas para un usuario, aumentan la seguridad y proporcionan un único lugar para guardar la información de los usuarios.

9. Application Servers (AS): Se enfoca en la parte de la aplicación o lógica del negocio. Conceptualmente un sistema puede tener muchas capas; sin embargo, la arquitectura más popular es de tres capas: interfaz, aplicación y base de datos.

1.5 Arquitectura Orientada a Servicios (SOA)

Los sistemas informáticos son fundamentales para los negocios modernos. Cada vez que el desarrollo de un sistema se interrumpe por cualquier causa, genera grandes pérdidas para la empresa, los clientes no quedan satisfechos y la misma pierde prestigios y confiabilidad para los clientes, lo cual es muy importante en el mundo del negocio. Esto sucedía antes porque las empresas creadoras de sistemas, diseñaban a estos como un solo proceso o servicio, entonces si fallaba algo por algún motivo en especifico, había que rediseñar e implementar todo de nuevo, pasando lo explicado anteriormente.

Por esta causa se creó la Arquitectura Orientada a Servicios (SOA por sus siglas en inglés), que es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio. Esta arquitectura permite dividir al gran sistema a desarrollar en pequeños servicios, bien definidos, para que en su conjunto resuelvan las peticiones de los usuarios.

1.5.1 ¿Qué es SOA?

A continuación se expone varios conceptos de SOA:

Según IBM:

Es un modelo de componente que interrelaciona unidades funcionales diferentes de una aplicación, denominado servicios, a través de interfaces y contratos bien definidos entre estos servicios. La interfaz se define de una manera neutral que debe ser independiente de la plataforma de hardware, del sistema operativo y del lenguaje de programación en los que se implemente el servicio. Esto permite que los servicios, construidos en una variedad de tales sistemas, interactúen entre sí de una manera uniforme y universal. [4]

Es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario. SOA permite la creación y cambios de los procesos de negocio desde la perspectiva de Tecnología Informática de forma ágil, a través de la composición de nuevos procesos, utilizando las funcionalidades de negocio que están contenidas en la infraestructura de aplicaciones actuales o futuras. [4]

SOA establece un marco de diseño para la integración de aplicaciones independientes de manera que, desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios. La forma más habitual de implementarla es mediante Servicios Web, una tecnología basada en estándares e independiente de la plataforma, con la que SOA puede descomponer aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular. [4]

En general SOA es una arquitectura que desglosa las funcionalidades principales del sistema en pequeños servicios para tener una mayor escalabilidad y disponibilidad, y esos servicios son accesibles a través de la red.

1.5.2 Elementos que conforman una SOA

- Proveedores de Servicios
- Consumidores de Servicios
- Bus Empresarial de Servicios

En realidad todo componente dentro de una organización puede ser tanto proveedor como consumidor de servicios debido a que el consumo de servicios en este tipo de arquitectura es jerárquico. Solo los

servicios de más bajo nivel son solo proveedores. Todos estos servicios interactúan entre sí a través de un Bus Empresarial de Servicios.

1.5.3 Beneficios de una SOA

Existen muchas razones por la cual es factible y enriquecedora la idea de utilizar una arquitectura SOA en una empresa. A continuación expondremos algunas características:

Reutilización: Es el factor fundamental en una arquitectura SOA. Las funciones de negocio se pueden definir como Web Services, y así se pueden reutilizar en otras necesidades de negocio.

Interoperabilidad: En una arquitectura débilmente acoplada, su objetivo es que la comunicación entre los clientes y los servicios sea independiente de la plataforma. Los protocolos que utiliza Web Services para su comunicación son independientes de la plataforma, el lenguaje de codificación y el sistema operativo, lo que facilita la comunicación con los socios del negocio.

Escalabilidad: Debido a que existe muy poca dependencia entre las aplicaciones clientes y los servicios que usan, las aplicaciones escalan muy fácilmente.

Flexibilidad: Es otra de las características que proporciona el débil acoplamiento entre los servicios. Siempre que se mantenga la interfaz de comunicación de un servicio, cualquier cambio dentro de este no afecta al resto.

Eficiencia de coste: Siempre que se habla de reutilización está implícita la eficiencia del coste, ya que no se invierte en lo que ya está realizado.

1.6 Herramientas de desarrollo

Java/Netbeans IDE

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsistems en la década de los noventa. Muchas estructuras de su sintaxis son tomadas de C y C++, pero tiene un modelo de objetos más simple. Lo que hace diferente a Java es que también es un entorno para la ejecución de programas desarrollados en Java, esto se hace gracias a la instalación de la Maquina Virtual de Java (JVM). La JVM presenta el Garbage Collection denominado comúnmente como recolector de basura. Dicho objeto se encarga de borrar los objetos creados que no se utilizan, para ahorrar o limpiar espacio en memoria.

Netbeans IDE es un Entorno de Desarrollo Integrador que soporta diferentes lenguajes (Ruby, C++, PHP, Java, etc.), principalmente Java. Fue desarrollado con el lenguaje Java y es de código abierto

(*open source*) por lo que se puede utilizar sin ninguna restricción. Además se le puede añadir o instalar componentes o plugins para facilitar el desarrollo del software. Presenta facilidades como entorno de desarrollo, completamiento de código y chequeo de sintaxis en tiempo real.

Erlang

Erlang es un lenguaje de programación funcional y concurrente. Apareció en los años 80 y fue desarrollado por la Empresa telefónica *Ericsson,* como un intento de desarrollar un lenguaje de alto nivel, estructurado y con capacidad de afrontar los proyectos que la empresa estaba desarrollando, especialmente los de la rama de Telecomunicaciones. Al principio era un lenguaje propietario de Ericsson, pero fue cedido como código abierto (*open source*) en 1998.

Este lenguaje presenta muchas características, las más relevantes son:

- Lenguaje de alto nivel basado en procesos.
- Acoplamiento de patrones (*Pattern-Matching*).
- Organización de memoria automática mediante el Colector de Basura (*Garbage Collector*).
- Presenta un sistema de ejecución que incluye una máquina virtual y librerías. El código de programa está formado por funciones, agrupadas en módulos.
- Su código secuencial es un lenguaje funcional con evaluación estricta, asignación única (las variables no se pueden sobrescribir para evitar efectos colaterales), y tipado dinámico.
- Proporciona el cambio en caliente de código, de forma que, éste se puede cambiar sin parar el sistema. Fue diseñado para realizar aplicaciones distribuidas, tolerantes a fallos, con respuestas en tiempo real y de funcionamiento ininterrumpido.
- La creación y gestión de procesos en Erlang es trivial, mientras que, en muchos lenguajes, los llamados hilos se consideran un apartado complicado y propenso a errores.
- La mayor fortaleza de Erlang es el soporte para concurrencia. Tiene un pequeño pero potente conjunto de primitivas para crear procesos y la comunicación entre los mismos. Dichos procesos son la forma principal de estructurar una aplicación, y se puede crear un gran número de ellos sin que se degrade el rendimiento.

- El soporte para procesos distribuidos es también parte de Erlang. Los procesos se pueden crear en nodos remotos, y la comunicación con ellos es transparente. Es decir, la comunicación con procesos remotos se hace exactamente de la misma manera que la comunicación con procesos locales.

En general, Erlang es en un lenguaje declarativo que representa una elevada potencia de computación y optimización de código, que al estar más próximo al razonamiento humano, se obtienen códigos más compactos, eficientes y de más fácil comprensión para el ser humano.

1.7 Protocolos de comunicación

XMPP

El Extensible Messaging and Presence Protocol (XMPP) es una tecnología abierta para la comunicación en tiempo real. Fue implementado por Jeremie Miller en el año 1998 con el nombre de Jabber, como formato para el intercambio de información utiliza XML, y además provee el intercambio de pequeñas piezas XML de una entidad a otra.

El protocolo es usado en una amplia gama de aplicaciones y es muy útil porque tiene un alto nivel de aplicaciones y servicios definidos por Internet Engineering Task Force (IETF) y XMPP Standards Foundation.

Un servicio, en este contexto, es una característica o función que puede ser utilizada por cualquier aplicación. Algunos de esos servicios son:

Codificación de canal:

Provee la encriptación entre un cliente y el servidor, también entre dos servidores. Es muy importante para la construcción de aplicaciones seguras.

Autenticación:

El servicio de autenticación también sirve para el desarrollo de aplicaciones seguras. Asegura que las entidades que intentan comunicarse a través de la red son autenticadas por un servidor.

Presencia:

Permite conocer la disponibilidad de otras entidades en la red.

Listas de contactos:

Permite el almacenamiento de una lista de contactos. Normalmente una entidad puede utilizar el servicio para tener una lista de contactos de entidades de confianza.

Mensajería one-to-one:

Permite que una entidad mande mensajes a otra entidad.

Mensajería multipartidista:

Permite unirse a una sala virtual para el intercambio de mensajería entre varias entidades.

Sesiones de medios de comunicación peer-to-peer:

Permite negociar y gestionar una sesión de chat con otra entidad. Tal sesión se puede utilizar para el chat de voz, video chat, transferencia de archivos, entre otros propósitos.

Cuando contamos con estos servicios y otros que no mencionamos, podemos desarrollar distintos tipos de aplicaciones:

- *Mensajería instantánea*
- *Groupchat*
- *Juegos*
- *Sistemas de control*
- *Geolocalización*
- *Middleware and cloud computing*
- *Sindicación de datos*
- *Servicios de identidad*

Desde hace varios años este protocolo se viene desarrollando y perfeccionando, y actualmente son muchas las empresas que desarrollan sus sistemas con este protocolo.

A continuación les mostraremos una pieza XML del servicio *Presencia*.

```
<presence
    from="Alejandro@simulando.prueba"
    to="SimulandoBD@simulando.prueba"
    type="subscribe"/>
```

Facebook es un sitio social que es famoso a nievel mundial gracias a la cantidad de usuarios que lo utilizan. A partir de febrero del 2010 este sitio incorporó el soporte Jabber/XMPP en su mensajería instantánea. Google también utiliza este protocolo en su mensajería instantánea Google Talk. Yahoo es otro que ha incursionado en la utilización del protocolo XMPP.

AMQP

Advanced Message Queuing Protocol (AMQP) es una tecnología todavía joven que está en desarrollo y en perfeccionamiento. La última versión es la 1.0.

El Grupo de Trabajo de AMQP está trabajando en las especificaciones de la infraestructura de mensajería para que ofrezca una sencilla pero potente manera de conectar las aplicaciones de mensajería.

La infraestructura de mensajería AMQP será:

- En general aplicable para el uso empresarial.
- Totalmente abierta.
- Plataforma agnóstica.
- Interoperable

AMQP permite la interoperabilidad completa para un middleware de mensajería, tanto el protocolo de red y la semántica de los servicios se definen en el AMQP.

Microsoft Corporation, Red Hat Inc, Progress Software, Cisco Systems, entre otras muchas empresas de gran nivel están contribuyendo en el desarrollo de este protocolo.

1.8 Metodologías de desarrollo de software

RUP

El Proceso Unificado de Racional (RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML) es la metodología más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

RUP es el resultado de varios años de desarrollo y de prácticas, donde se unifican técnicas de desarrollo y trabajo de muchas metodologías utilizadas por los clientes.

Define como principales elementos:

- **Trabajadores (“quién”):** Define el rol de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- **Actividades (“cómo”):** Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos
- **Artefactos (”qué”):** Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- **Flujo de actividades (“Cuándo”):** Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

RUP agrupa las actividades definiendo 9 flujos de trabajo como se muestra en la figura:

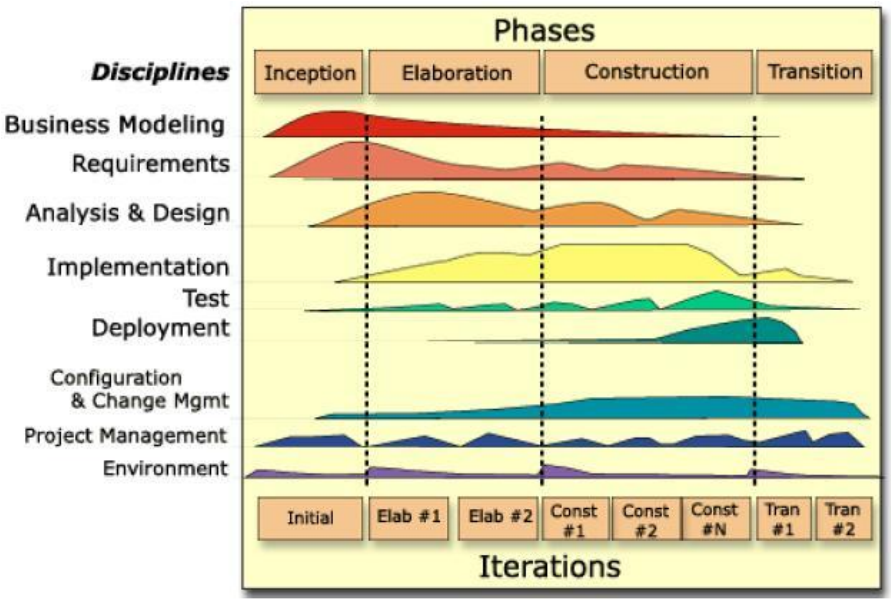


Figura 7. Proceso Unificado Racional.

Los 6 primeros son flujos de ingeniería y los otros 3 son de apoyo. En cada flujo se realizan actividades definidas y los trabajadores producen y consumen artefactos definidos dentro de cada flujo. Como pueden ver en la figura se definen 4 fases, que cada una representa un estado del producto y produce un hito que sirve de entrada para la próxima fase. Todos los flujos están presentas en las 4 fases, pero en dependencia de la fase en que se encuentre el producto, hay flujos que tienen más cargas que otros.

El ciclo de vida de RUP se caracteriza por ser dirigido por casos de uso, que reflejan lo que los usuarios futuros necesitan y desean, guiando el proceso de desarrollo ya que los demás artefactos representan la realización de los casos de uso; centrado en la arquitectura que muestra la visión común del sistema completo y describe los elementos del modelo que son más importantes para su construcción; iterativo e Incremental, lo que significa que cada fase se desarrolla en iteraciones que involucran actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros, obteniendo un producto con un determinado nivel que irá creciendo incrementalmente en cada iteración.

XP

Basándose en la simplicidad, la comunicación y la reutilización del código, surge la Programación Extrema (XP), siendo así una metodología ligera de desarrollo de software. XP consiste en una programación extrema (rápida). Como requisito para alcanzar el éxito del proyecto se tiene al usuario final como parte del equipo. Es una metodología con reconocido éxito y se utiliza en proyectos con entregas a cortos plazos.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Lo esencial en este proceso de desarrollo es lograr la comunicación entre desarrolladores y usuarios, la retroalimentación entre ellos y con los usuarios finales y la simplicidad en el código.

SCRUM

SCRUM define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprint, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo del proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

Aunque presenta otra serie de características como:

- Equipos auto dirigidos.

- Utiliza reglas para crear un entorno ágil de administración de proyectos.
- No prescribe prácticas específicas de ingeniería.
- Los requerimientos se capturan como ítems de la lista reserva del producto.
- El producto se construye en una serie de sprint de un mes de duración.
- Usado para proyectos complejos con requerimientos cambiantes.
- Basado en un control de proceso empírico.

Prácticas:

- Iteración (Sprint).
- Reunión de planificación de las iteraciones.
- Reuniones diarias.
- Reuniones de revisión de las iteraciones.
- Reuniones de revisión del diseño.
- Estabilización de las iteraciones.

1.9 Herramientas CASE

Visual Paradigm

Visual Paradigm es una herramienta CASE de diseño que utiliza “UML”: como lenguaje de modelado. Es una herramienta muy potente y de fácil uso. Te permite dibujar todo tipo de diagramas UML, revertir código fuente a modelos UML, generar código fuente desde los diagramas UML. Además generar documentación automáticamente en varios formatos como html, pdf o doc y permite el control de versiones. Por otro lado la herramienta es colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto. Un detalle importante es que es una herramienta libre y gratuita.

Rational Rose

Rational Rose Enterprise es el producto más completo de la familia Rational Rose. Todos los productos Rational Rose incluyen soporte UML. Es una herramienta case muy potente pero es software propietario y no es para nada barato, y como la UCI desarrolla y utiliza software libre, no se utiliza.

Es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. [5]

Esta herramienta propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

Rational Rose utiliza un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Cada iteración comienza con una primera aproximación del análisis, diseño e implementación para identificar los riesgos del diseño, los cuales se utilizan para conducir la iteración, primero se identifican los riesgos y después se prueba la aplicación para que éstos se hagan mínimos.

Cuando la implementación pasa todas las pruebas que se determinan en el proceso, ésta se revisa y se añaden los elementos modificados al modelo de análisis y diseño. Una vez que la actualización del modelo se ha modificado, se realiza la siguiente iteración.

1.10 Selección de herramientas, protocolos y metodología de desarrollo.

Como metodología de desarrollo se ha seleccionado XP debido a que encaja perfectamente con el tipo de proyecto, las condiciones y con la idea de desarrollo que se tiene del sistema. A continuación, las razones fundamentales que se tuvieron en cuenta al escoger esta metodología.

- El proyecto es pequeño. XP está concebida para ser utilizada dentro de proyectos pequeños y de desarrollo rápido se adapta perfectamente a este caso.
- Pocos roles. Esta metodología está dirigida a grupos de desarrollo pequeños y con pocos roles como este caso.

- El manejo del cambio se convierte en parte sustantiva del proceso. A medida que el proyecto avanza pueden surgir nuevas expectativas o ideas que pueden ser incorporadas fácilmente permitiéndole mayor adaptabilidad al producto, con la metodología XP esto es completamente factible pues esta se adapta perfectamente a los proyectos cuyos requerimientos cambian a menudo.

- El cliente o el usuario se convierten en miembro del equipo. Con el uso de esta metodología y la importancia que esta le concede a la retroalimentación, el cliente es parte del equipo de desarrollo y en este caso que se desarrolla un proyecto para desarrolladores la relación es aún más fuerte.

- Propiedad colectiva del código. XP plantea que todos los programadores pueden realizar cambios en cualquier parte del código en cualquier momento. En el proceso de desarrollo con que cuenta la empresa esta es una práctica común.

- Comunicación de los programadores a través del código. XP enfatiza el uso de líneas directivas para la codificación que están bien establecidas. Desde sus comienzos la empresa cuenta con una línea directiva para la codificación.

Se decide por el lenguaje Erlang para desarrollar el middleware de comunicaciones entre los servidores, para lanzarlo en la Universidad de las Ciencias Informáticas como la mejor opción para el desarrollo de aplicaciones de red, sobre todo para el desarrollo de aplicaciones servidoras. Y se selecciona al lenguaje Java para el desarrollo del API de comunicaciones a usar por los clientes para comunicarse con la Nube.

El protocolo que se va a usar es XMPP porque a diferencia de AMQP está, desde hace varios años, en desarrollo y perfeccionándose. Actualmente existen muchas aplicaciones de mensajería que utilizan XMPP como Google Talk, la mensajería instantánea de Facebook, entre otras. Además, como AMQP es muy joven todavía no existe mucha documentación sobre este protocolo.

Como herramienta case se decide por Visual Paradigm UML 6.4 Community Edition, porque es alternativa libre y gratuita, en comparación con Rational Rose que es propietario.

1.11 Conclusiones Parciales

En este capítulo se ha hecho un estudio de varios conceptos como Cloud Computing, Sistemas distribuidos, SOA y Middleware para poder desarrollar la infraestructura de comunicación.

Se define que Cloud Computing es un concepto que permite el acceso a aplicaciones que están en servidores en la Nube y que engloba los conceptos de SaaS, PaaS, HaaS, entre otros.

Los sistemas distribuidos son aquellos sistemas que sus funcionalidades o módulos están distribuidas en una red y que resuelven una tarea en común. Estos sistemas distribuidos presentan características como: concurrencia, carencia de reloj global, compartición de recursos, apertura, escalabilidad, tolerancia a fallos, transparencia, entre otras.

Middleware es un subsistema de los sistemas distribuidos que se encarga de gestionar los mensajes de un componente a otro. Existen varios tipos de Middleware: Distributed Tuples (DT), Remote Procedure Call (RPC), Messaging Oriented Middleware (MOM), Distributed Object Middleware (DOM), Common Object Request Broker Arquitecture (CORBA), entre otros.

SOA en general es una arquitectura que desglosa las funcionalidades principales del sistema en pequeños servicios para tener una mayor escalabilidad y disponibilidad, y esos servicios son accesibles a través de la red. Los beneficios de una SOA son: Reutilización, Interoperabilidad, Escalabilidad, Flexibilidad y Eficiencia de coste.

También en este capítulo se estudia los lenguajes Erlang y Java, y como IDE de desarrollo de Java tenemos a Netbeans, escogiendo Erlang para el desarrollo del middleware de comunicaciones entre los servidores que conforman la Nube. Como protocolo de comunicación se hace un análisis de XMPP y AMQP, escogiendo a XMPP porque lleva muchos años en perfeccionamiento a diferencia de AMQP que todavía es joven. Dentro de las metodologías de desarrollo que se analizan están, RUP, XP y SCRUM, y se escoge a XP ya que este tipo de proyecto es sencillo y rápido. Por último se analiza como herramientas case a Rational Rose y a Visual Paradigm y se escoge a Visual Paradigm debido a que es software libre a diferencia de Rational Rose que es software propietario.

CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

El sistema a desarrollar es una infraestructura que debe permitir la comunicación entre las aplicaciones que estén desplegadas en la plataforma con el modelo Cloud Computing. Utilizando el lenguaje Erlang y el protocolo de comunicación XMPP, podemos desarrollar un middleware de tipo MOM (Messaging Oriented Middleware). Los middleware están caracterizados como infraestructuras, por tanto, desarrollando el middleware estaremos obteniendo la infraestructura propuesta en este trabajo de diploma.

2.2 Requerimientos no funcionales

Los requerimientos no funcionales de un sistema son propiedades o cualidades, los cuales se usan como características que hacen que el producto sea atractivo, usable, rápido o confiable.

Diseño e implementación:

- Middleware de comunicación escrito en Erlang
- API de comunicación que utilizan los clientes escrito en Java
- El protocolo de comunicación que se usa es el XMPP
- Se utiliza como metodología de desarrollo XP

Factibilidad:

Costo de desarrollo: Se utiliza lenguajes de libre distribución y bajo costo de implementación.

Esfuerzo: Se requiere poco personal y esfuerzo utilizando tecnologías modernas y lenguajes de programación de alto nivel, los cuales reducen considerablemente la cantidad de líneas de código necesarias para desarrollar aplicaciones; basadas en el paradigma de programación funcional.

Tiempo de desarrollo: Al utilizar tecnologías de alto nivel; significando esto que la programación se realiza en un lenguaje más cercano al razonamiento humano, se hacen posible la implementación de sistemas en un tiempo relativamente corto.

Rendimiento:

El sistema debe ser eficiente al 100% en cuanto al envió de mensajes entre los nodos que se comunican.

Hardware:

Los procesos que se desarrollan con el lenguaje Erlang no se llevan mucho recurso. Se han llegado a probar 1 millón de procesos en una PC y ha seguido cumpliendo con las tareas sin trabas y sin lentitud. Con esta opción que brinda esta tecnología, el sistema a desarrollar no requiere mucho hardware, solo depende de las aplicaciones que estén integradas al sistema.

Disponibilidad:

El sistema debe tener un 100% de disponibilidad para que funcione las 24 horas del día y así haya comunicación siempre.

2.3 Cloud Computing y las condiciones técnicas de la UCI y de Cuba

La red nacional de nuestro país presenta muchos problemas de conectividad y rapidez, debido a que no hay recursos para crear buenas condiciones en la red. Pero esta problemática situación no es un obstáculo para poder desarrollar una plataforma con modelo Cloud Computing. Existen experiencias con aplicaciones con servicios de mensajería que utilizan el protocolo XMPP. El Jabber utiliza dicho protocolo y es utilizado en distintos lugares geográficos del país, mayormente es utilizado en la Universidad de las Ciencias Informáticas (UCI). Utilizando el Jabber un estudiante o profesor se puede comunicar con otra persona en cualquiera de las facultades regionales que pertenecen a la UCI, ubicadas en el occidente, centro y oriente del país.

Con lo explicado anteriormente podemos concluir de que se puede desarrollar este trabajo de diploma cuyo objetivo es lograr una infraestructura de comunicación para una plataforma con modelo Cloud Computing.

2.4 Comportamientos OTP (OTP Behaviors)

Los comportamientos OTP son una formalización de los patrones de diseño para el trabajo con procesos. La formalización se puede llevar a cabo por módulos que existen en la librería que proporciona la distribución estándar de Erlang/OTP. Estos módulos pueden realizar la labor de procesos de trabajo genéricos y el manejo de errores. El código escrito por el programador es ubicado en un módulo aparte, y es consultado a través de funciones predefinidas de devolución de llamadas.

En este ambiente existen los procesos *workers* (trabajadores) y *supervisors* (supervisores), donde los trabajadores realizan el proceso real y los supervisores supervisan los procesos trabajadores y a otros procesos supervisores. Los procesos trabajadores encapsula a los comportamientos servidores (*gen_server*), controladores de eventos (*gen_event*) y maquinas de estado finito (*gen_fsm*) y los supervisores al comportamiento supervisor (*supervisor*). Con esta estructura se crea lo que se denomina el árbol de supervisión, como se muestra en la figura 8.

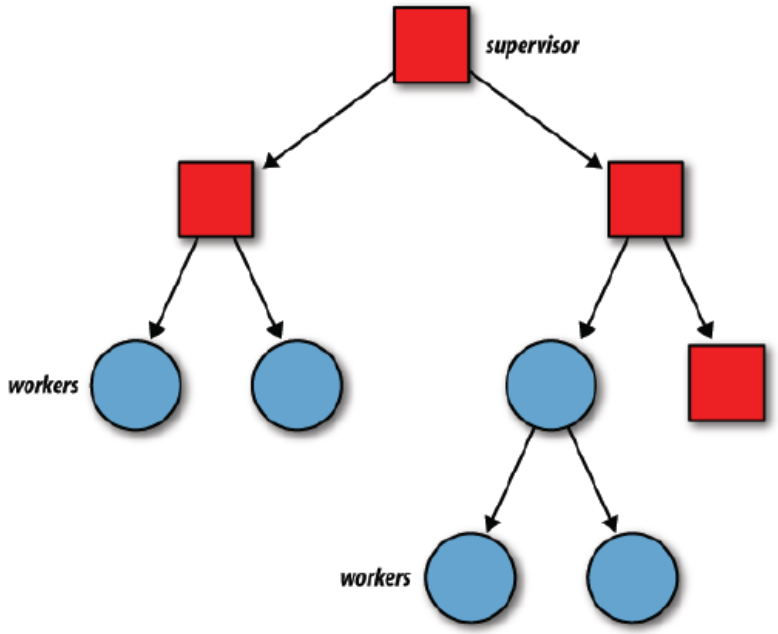


Figura 8. Árbol de supervisión de procesos

Los árboles de supervisión son empaquetados en el comportamiento aplicación (*application*). Una aplicación OTP es un componente que realiza determinadas tareas de un tema específico, y no se debe confundir con el concepto más general de una aplicación.

2.5 Conclusiones parciales

En la introducción de este capítulo se explica algo muy importante que hay que tener en cuenta de cuando se habla de infraestructura en este trabajo de diploma. Al utilizar el lenguaje Erlang y el protocolo de comunicación XMPP, podemos desarrollar un middleware de tipo MOM (Messaging Oriented Middleware), y que un middleware está caracterizado como una infraestructura.

También, en este capitulo, se habla de los requisitos funcionales en cuanto a diseño e implementación, factibilidad, rendimiento y disponibilidad. En cuanto a la arquitectura de dicho sistema a desarrollar, esta basado en los comportamientos OTP que incluye el lenguaje Erlang. En este capitulo se ha reservado un epígrafe para hablar de dichos comportamientos, donde de los cuales vamos a utilizar los comportamientos de servidores, supervisores y aplicación.

EXPLORACIÓN Y PLANIFICACIÓN DEL SISTEMA

3.1 Introducción

En el presente capítulo se hace alusión a las fases de exploración y planificación propias de la metodología de desarrollo utilizada para la implementación del sistema que se propone. Se exponen además los artefactos generados durante el transcurso de las mismas.

3.2 Fase de exploración. Definición

En esta fase se exploran las herramientas y tecnologías a usar, las diferentes formas de resolver problemas concretos de implementación y también se definen historias de usuario, que son la forma de definir los requisitos del sistema a implementar.

En el sistema no se definieron casos de usos y requisitos funcionales, que de alguna u otra forma están relacionados con las historia de usuario, debido a que dicho sistema solo tiene la tarea de hacer de puente para la comunicación entre varias aplicaciones y los clientes que consumen los servicios de esas aplicaciones, mediante el protocolo XMPP. Con la ayuda de las herramientas y tecnologías, que se van a usar para desarrollar la infraestructura, se hace fácil la realización del sistema. Por lo planteado anteriormente se define suplantar a las historias de usuario por los componentes que conforman al sistema como tal.

3.2.1 Componentes

Nodo:

Este componente está encargado de recibir los datos que le son enviados a la aplicación que va a estar en su entorno de trabajo, y enviar los datos que dicha aplicación desee enviar. En todo este proceso estará inmerso el protocolo XMPP. Su prioridad de negocio es de nivel medio.

Controlador:

La tarea del controlador es controlador los nodos que están en su red local y los datos que se reciben y se envían pasan a través de él. Su prioridad de negocio es de nivel medio.

Centro de Control:

La tarea de este componente es tener el dominio de los controladores que existen en la red. Además, hace de interface de comunicación entre los clientes y la plataforma que va a estar desplegada en la Nube. Su prioridad de negocio es de nivel medio.

ConexionAPI:

Este componente es una librería para poder comunicarse con la plataforma con el modelo Cloud Computing. Su prioridad de negocio es de nivel medio.

AplicacionNubeAPI:

Este componente es una librería para que una aplicación pueda desplegar sus servicios en la nube. Su prioridad de negocio es de nivel medio.

GestionXML:

En cada uno de los componentes Nodo, Controlador y Centro de Control va a haber una aplicación OTP que va a brindar los servicios para trabajar con los datos de un XML que representa la comunicación con el protocolo XMPP. Este presenta una prioridad de negocio alta debido a que es la aplicación encargada de interpretar el protocolo XMPP.

3.3 Estimación y planificación. Definición

En esta fase se priorizan componentes del sistema y se acuerda el alcance de la planificación de entrega (*release*). Se estima cuánto esfuerzo requiere cada componente y a partir de allí se define el

cronograma. Esta estimación se expresa utilizando como medida el punto. Un punto se considera como una semana ideal de trabajo donde se trabaja el tiempo planeado sin ningún tipo de interrupción.

La planificación no debe ser estricta puesto que hay muchas variables en juego, debe ser flexible para poder adaptarse a los cambios que puedan surgir. Una buena estrategia es hacer planificaciones detalladas para unas pocas semanas y planificaciones mucho más abiertas para unos pocos meses.

3.3.1 Estimación de esfuerzo por componentes

La estimación de esfuerzo que se decidió para el desarrollo de cada uno de los componentes se representa en la siguiente tabla:

Tabla 3.1: Plan de estimación de esfuerzo por componentes.

Componentes	Puntos estimados
Nodo	0.2
Controlador	0.2
Centro de Control	0.2
ConexionAPI	0.2
AplicacionNubeAPI	0.2
GestionXML	1

3.3.2 Plan de iteraciones

La metodología XP contiene la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas [6].

La etapa de desarrollo se ha decidido realizarla en solo dos iteraciones, teniendo en cuenta la prioridad de cada componente, además de la facilidad y agilidad que brinda las tecnologías que se van a usar, para el desarrollo de sistemas. Cada componente ofrece tareas específicas de programación. Así mismo, para cada uno se le hará las pruebas de aceptación. Estas serán pequeñas pruebas que se realizan al final del ciclo en el que se desarrollan, para verificar que cada uno de los componentes funcione correctamente y que no afecte el funcionamiento de los otros.

Iteración 1:

En esta iteración se implementa los componentes de mayor prioridad, que sería la aplicación OTP GestionXML. Dando los primeros resultados y los más importantes del desarrollo de la infraestructura.

Iteración 2:

En esta segunda iteración se implementará los componentes que quedan, siendo los de menor prioridad y los más fáciles de implementar. Dentro de ese dominio están los componentes Nodo, Controlador, Centro de Control y las librerias ConexionAP y AplicacionNubeAPI. Al final de esta iteración se obtendrá la versión 1.0 final del producto. Esta versión se pondrá en funcionamiento para la evaluación de su comportamiento y funcionamiento.

3.3.3 Duración de las iteraciones

Para una mayor organización del trabajo como lo plantea el ciclo de vida de XP se crea un plan de duración de las iteraciones, en este caso se realizaría un solo plan ya que existe un único equipo de desarrolladores. A continuación se muestra cómo estarán organizados los componentes según el orden que serán abordados en cada iteración según su prioridad y el tiempo de duración de las mismas.

Tabla 3.2: Plan de iteraciones.

Iteraciones	Orden de los componentes a implementar	Duración total de la iteración
Iteración 1	1. GestionXML	1 semana
Iteración 2	2. Nodo 3. Controlador 4. Centro de Control 5. ConexionAPI 6. AplicacionNubeAPI	1 semana

3.3.4 Plan de entregas

En el plan de entrega que se plantea a continuación se hace una propuesta de la fecha aproximada en que se harán versiones (*releases*) al sistema al finalizar cada iteración en la fase de implementación.

Tabla 3.3: Plan de entregas.

Entregable	Final 1ra iteración primera semana de abril	Final 2da iteración cuarta semana de abril
Infraestructura de comunicaciones para una plataforma de Cloud Computing	0.1	1.0

3.4 Conclusiones parciales

En este capítulo se abordó la planificación para el desarrollo de la infraestructura de comunicación. Se tomo como acuerdo cambiar las historias de usuarios por los componentes que conforman al sistema. También se explica el plan y la duración de cada una de las iteraciones además del plan de entrega que se va a llevar a cabo.

DISEÑO, IMPLEMENTACIÓN Y PRUEBA

4.1 Introducción

En este capítulo vamos a ver el diseño de todo el sistema y como se van a integrar los componentes que conforman al sistema. También veremos la implementación, las pruebas y la validación de dicho sistema.

4.2 Diseño del sistema

Para el diseño del sistema se tuvo en cuanta las herramientas y tecnologías que se van a usar en la implementación de dicho sistema. La infraestructura de comunicación está compuesta por los componentes Nodo, Controlador, centro de Control, ConexionAPI, AplicacionNubeAPI y GestionXML.

Los clientes que se conectan con la Nube para consumir los servicios de las aplicaciones desplegadas, lo hacen a través de la librería implementada en Java ConexionAPI. Esta librería se conecta con el componente Centro de Control vía socket. El Centro de Control además de gestionar la comunicación con los clientes, se encarga de gestionar el envió de piezas XML, así como su registro, y controla a todos los Controladores que existen por cada red local. Los Controladores controlan a todas las aplicaciones que existen en su red local, además gestionan el envió y recibimiento de las piezas de esas aplicaciones a través del componente Nodo. El componente Nodo permite la comunicación de la aplicación con el Controlador de su red local. Para que la aplicación pueda desplegar sus servicios en la Nube, debe utilizar la librería AplicacionNubeAPI, la cual permite la comunicación con el componente Nodo. (***Ver anexos 2 y 3 para una mejor observación***)

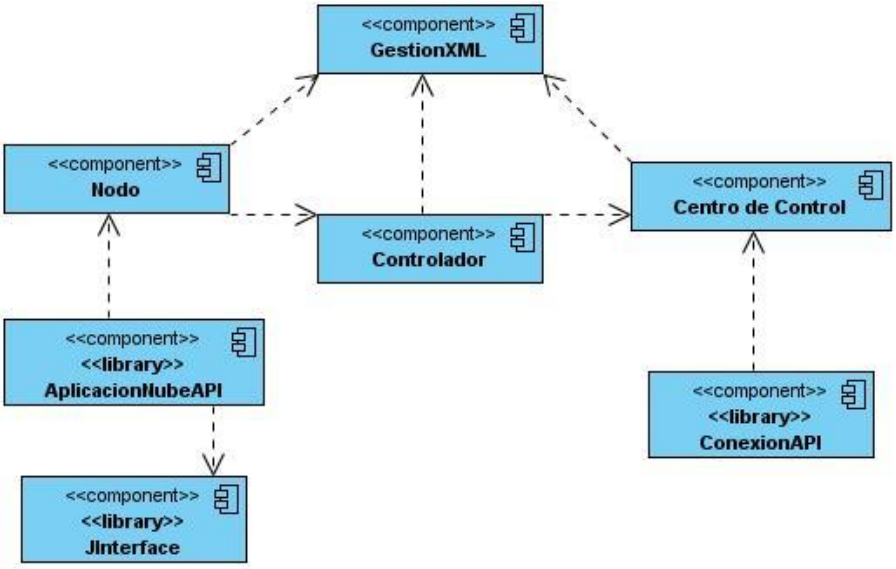


Figura 9: Diagrama de componentes del sistema

Como el lenguaje que se utiliza no es orientado a objeto, es un lenguaje funcional, para poder obtener los artefactos en la parte del diseño, se tuvo en cuenta la siguiente conversión:

Tabla 4.1: Conversión del diseño de lenguaje Erlang a OO.

Lenguaje Orientado a Objetos (OO)	Lenguaje funcional Erlang
Objeto	Proceso
Clase	Módulo
Métodos	Funciones
Atributos	Parámetros de los procesos

4.2.1 Diseño de la aplicación OTP GestionXML

Este componente se va a desarrollar con los comportamientos (*behaviors*) aplicación, supervisor y servidor. El módulo GestionXML estructurado con el comportamiento servidor, va a brindar los servicios que se muestran en la figura 9.

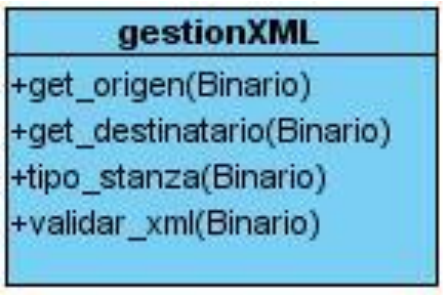


Figura 10: Servicios que brinda el módulo GestionXML

4.2.2 Diseño del componente Nodo

El componente Nodo va a estar desplegado en cada ordenador que contenga una aplicación que brinde servicios en la Nube para mantener la comunicación de la aplicación con el componente Controlador. Consta de dos módulos, *gestionNodo* e *i_comunicacion*, más el componente GestionXML para poder obtener la información necesaria de cada pieza XML que se envíe ó se reciba. El módulo *gestionNodo* realiza todas las operaciones de mayor peso en este componente, enviar y recibir la pieza en su expresión binaria. El módulo *i_comunicacion* es solo la interface para la comunicación con la aplicación, la cual recibe de la aplicación el URL de la pieza XML para el envío y cada vez que la aplicación recibe un envío esta interface le brinda el URL de la ubicación de la pieza.

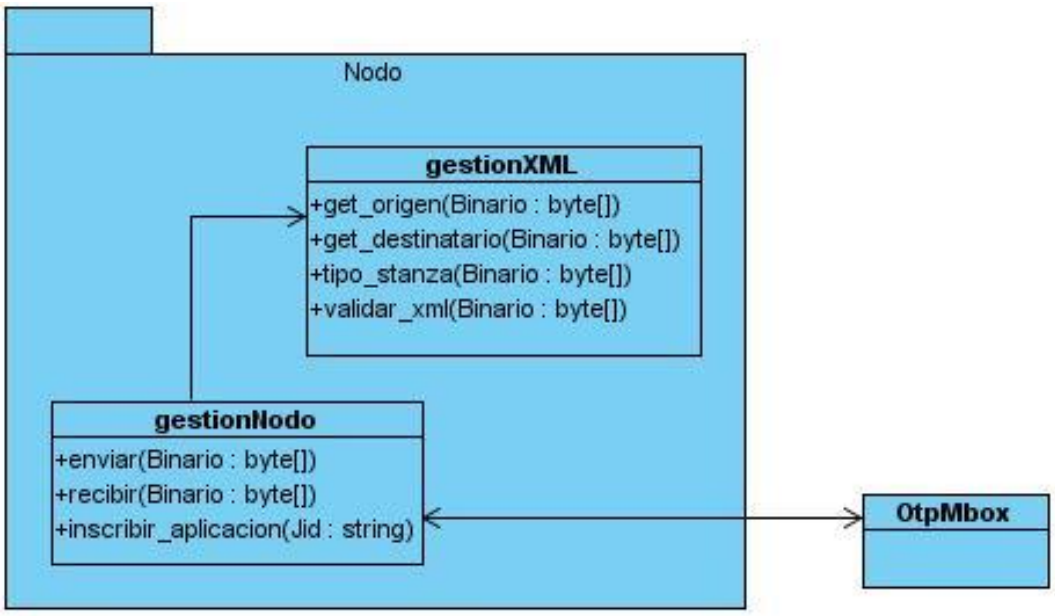


Figura 11: Diagrama del diseño del componente Nodo.

4.2.3 Diseño del componente Controlador

Este componente solo está compuesto por un solo módulo llamado *gestionControlador*, el cual utiliza también el componente GestionXML. Este componente Controlador gestiona el envió y recibimiento de piezas XML de todas las aplicaciones dentro de una red local, a través del componente Nodo, por lo que por cada red local que exista en la Nube va a existir un controlador. Por cada pieza XML que se envíe este módulo levanta un proceso cuya tarea es la de verificar si el envío llego a su destino, sino es así este proceso notifica al proceso *gestionControlador* para buscar su destino en el Centro de Control.

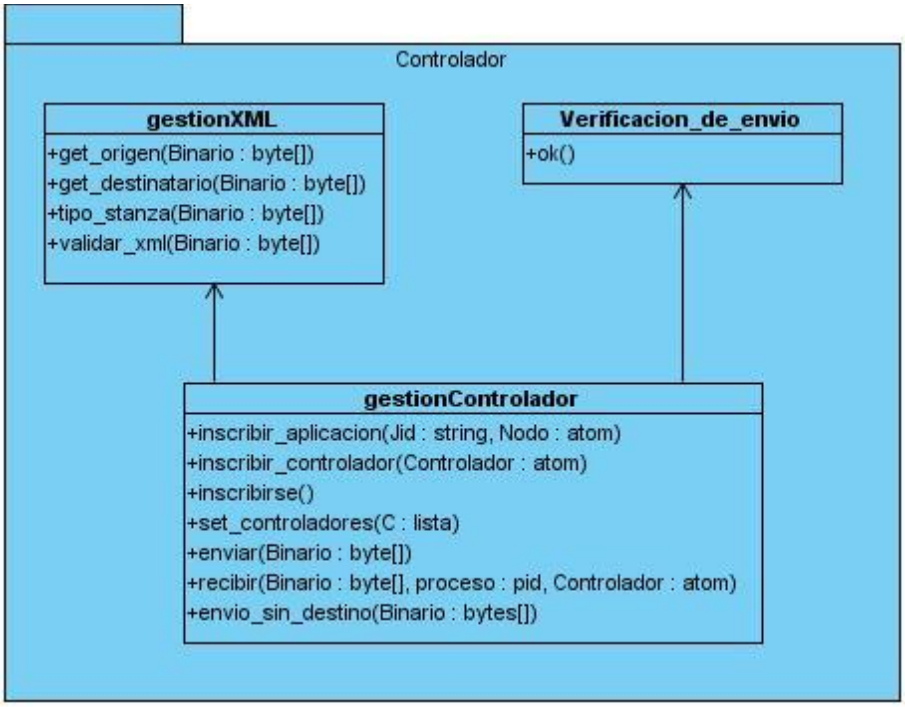


Figura 12: Diagrama del diseño del componente Controlador.

4.2.4 Diseño del componente Centro de Control

El Centro de Control se encarga de controlar a todos los Controladores que existen en la Nube y además es el encargado de que los clientes que van a consumir los servicios de las aplicaciones que están desplegadas en la Nube, se comuniquen con el sistema.

El componente está compuesto por el módulo *centro_de_control*, el cual se encarga de gestionar todo el tránsito de las piezas XML, además de registrar el envío. El *socket* es otro módulo en este componente que se encarga de mantener la comunicación con los clientes vía socket. En este

componente también interviene el componente GestionXML para poder obtener la información necesaria de cada pieza XML.

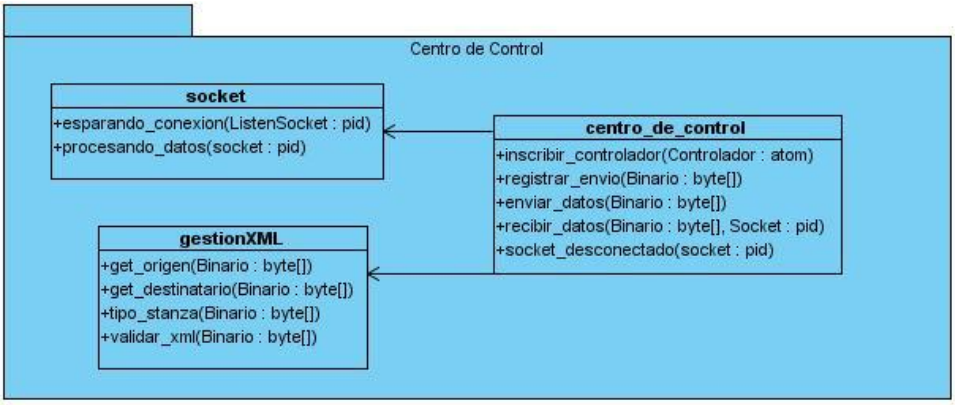


Figura 13: Diagrama del diseño del componente Centro de Control

4.2.5 Diseño del componente ConexionAPI

Este componente es una librería implementada con el lenguaje orientado a objetos Java, para que los clientes, usando esta librería, se puedan comunicar con la Nube.

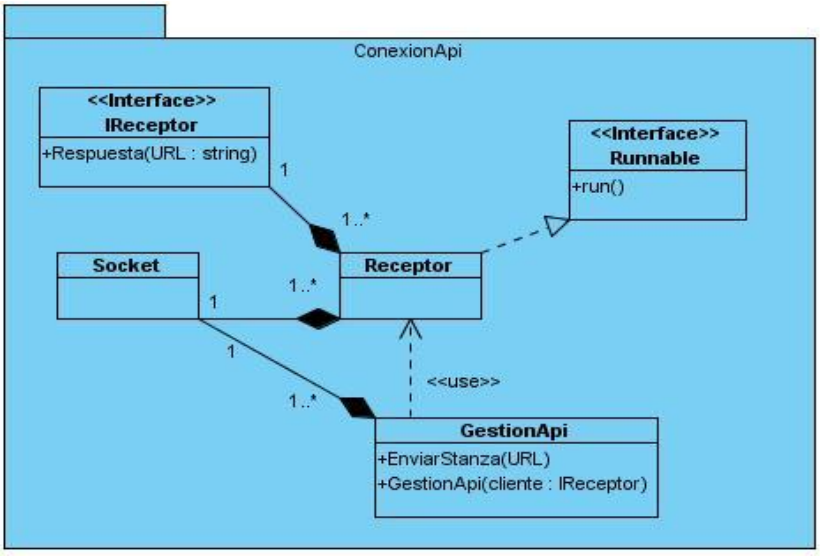


Figura 14: Diagrama del diseño de clases del componente ConexionAPI

4.2.6 Diseño del componente AplicacionNubeAPI

Este componente es una librería implementada con el lenguaje orientado a objetos Java, para que las aplicaciones desarrolladas con este mismo lenguaje puedan desplegar sus servicios en la Nube.

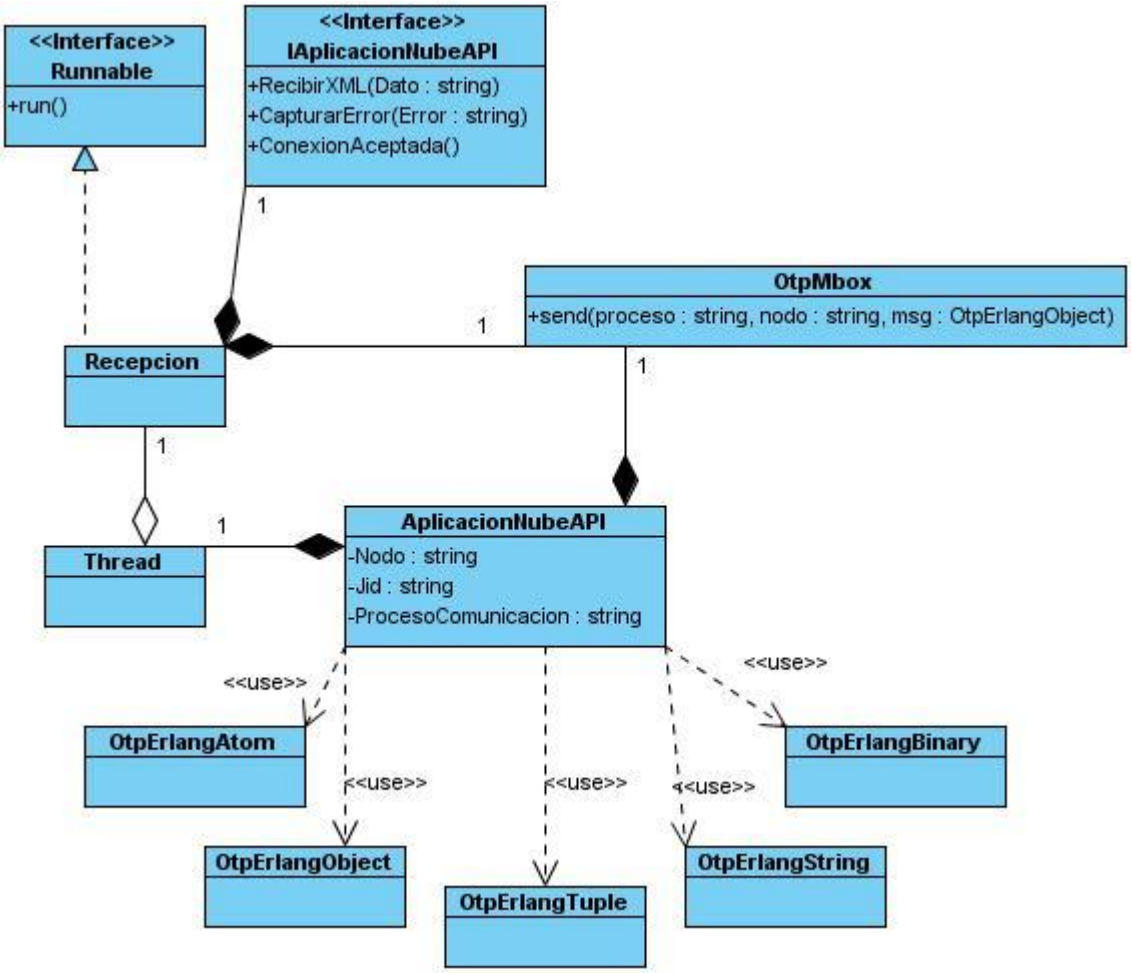


Figura 15: Diagrama del diseño de clases del componente AplicacionNubeAPI

4.3 Implementación del sistema

El desarrollo es la parte más importante en el proceso de la programación extrema. Todos los trabajos tienen como objetivo que se programen lo más rápidamente posible, sin interrupciones y en dirección correcta. También es muy importante el diseño, y se establecen los mecanismos, para que éste sea revisado y mejorado de manera continuada a lo largo del proyecto, según se van añadiendo funcionalidades al mismo [7].

La metodología XP plantea una serie de prácticas básicas para una producción efectiva, estas deben seguirse al pie de la letra para que el proyecto tenga un desarrollo exitoso.

Estas se describen a continuación:

- **La planeación:** En la cual la opinión del cliente y del equipo de desarrollo deben fusionarse como un todo coherente. El cliente y los programadores negocian el alcance del proyecto para cada iteración, esta etapa es conocida como “Juego de Planificación”. El factor crítico es permitir al cliente tomar las decisiones de negocio y al equipo de desarrollo tomar las decisiones técnicas. (Ver figura 1 en los Anexos).

- **Entregas en iteraciones pequeñas:** Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.

- **Hacer historias y usar metáforas:** Guiar todo el desarrollo del sistema a través de una Historia Compartida por el Equipo (o Metáfora) acerca de cómo trabaja (o debería trabajar) el Sistema.

- **Diseñar simple:** El sistema debería diseñarse de la manera más simple posible en cualquier momento dado. La complejidad extra es removida, tan pronto como es descubierta.

- **Probar:** Los desarrolladores continuamente escriben pruebas unitarias, las cuales deben correr sin error para que el desarrollo pueda continuar. Cuando se detecta un error en una corrida, su reparación pasa a ser la máxima prioridad para el Programador y/o el Equipo. Los Clientes (ayudados por desarrolladores) participen en el diseño de pruebas funcionales para probar qué funcionalidades están terminadas de acuerdo a sus expectativas.

- **Refactorizar:** Los desarrolladores reestructuran el sistema sin cambiar su comportamiento para remover duplicación de código, mejorar la comunicación, simplificar el código, o agregar flexibilidad.

- **Programar por pares:** Todo el código desarrollado es escrito por dos desarrolladores sentados frente a una única estación de trabajo. El que utiliza el teclado piensa en la mejor manera de implementar alguna funcionalidad, el otro piensa estratégicamente, cuestionando si se puede simplificar, anticipando pruebas, o preguntándose si el enfoque es el adecuado o si debe descartarse código y replantear el problema. Se alienta la rotación continua de programadores en los pares, combinando diferentes niveles de experiencia y de conocimiento del código.

- **Propiedad colectiva:** Cualquier integrante del Equipo puede cambiar cualquier código de cualquier parte del sistema en cualquier momento. Se asume que al seguir las prácticas de XP,

después de un tiempo razonable, cualquier programador conoce todo el código, principalmente por el hecho de la programación en pares.

- **Integrar continuamente:** El sistema se integra y se construye (por ejemplo, se compila), es decir, se unen sus partes, varias veces por día, hasta el extremo de integrar el sistema completo, cada vez que se termina una tarea.

- **Semanas de 40 horas:** Trabajar no más de cuarenta horas por semana como una regla estándar. Nunca trabajar sobre-tiempo dos semanas seguidas; si esto es necesario, hay problemas más grandes que hay que descubrir.

- **Cliente disponible localmente:** Es condición esencial la inclusión de al menos un Cliente real, vivo, como parte del Equipo. Debe estar disponible en turnos completos para responder preguntas e interactuar con el resto del Equipo.

- **Usar estándares de codificación:** Los desarrolladores escribirán todo el código de acuerdo a reglas predeterminadas que enfatizarán la comunicación a través del código. Estos estándares serán simples de seguir y se seguirán siempre.

La forma iterativa para la implementación de un software que plantea XP junto a estas prácticas da como resultado que al culminar cada iteración se obtenga una versión del producto funcional, éste debe ser probado y mostrado al cliente sirviendo de retroalimentación para el equipo de trabajo, se exponen detalladamente las dos iteraciones generadas por la planificación descrita anteriormente así como las tareas que se plantearon para la realización de cada uno de los componentes y las pruebas que se efectuaron al sistema.

4.3.1 Iteración 1

En esta primera iteración se desarrollan las aplicaciones OTP, las cuales son las más importantes porque permiten el trabajo con el protocolo XMPP.

Tabla 4.2: Componentes tratados en la 1ra iteración.

No.	Componentes	Estimación	Real
1	GestionXML	1	1
Total		1	1

Tareas generadas por las Aplicaciones OTP:

Tabla 4.3: GestionXML. Tarea 1

Tarea	
Número de tarea: 1	**Número de componente:** 1
Nombre de la tarea: Diseñar e implementar la aplicación OTP GestionXML.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: Aplicación que va a permitir el trabajo con el protocolo XMPP	

4.3.2 Iteración 2

En esta última iteración se desarrollan los componentes que quedan, obteniendo una versión final del sistema.

Tabla 4.4: Componentes tratados en la 2da iteración.

No.	Componentes	Estimación	Real
2	Nodo	0.2	0.2
3	Controlador	0.2	0.2
4	Centro de Control	0.2	0.2
5	ConexionAPI	0.2	0.2
6	AplicacionNubeAPI	0.2	0.2
Total		1	1

Tareas generadas por cada componente:

Tabla 4.5: Componente "Nodo". Tarea 1

Tarea	
Número de tarea: 1	**Número de componente:** 2

Nombre de la tarea: Diseñar e implementar el módulo gestionNodo.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: El módulo gestionNodo permite toda la gestión que se hace en el componente Nodo, creando el proceso para dicha tarea.	

Tabla 4.6: Componente "Nodo". Tarea 2

Tarea	
Número de tarea: 2	**Número de componente:** 2
Nombre de la tarea: Diseñar e implementar la interface i_comunicacion.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: Interface que permite la comunicación de la aplicación desplegada en la plataforma con el middleware desarrollado.	

Tabla 4.7: Componente "Controlador". Tarea 1

Tarea	
Número de tarea: 1	**Número de componente:** 3
Nombre de la tarea: Diseñar e implementar el módulo gestionControlador.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: El módulo gestionControlador permite toda la gestión que se hace en el componente Controlador,	

creando el proceso para dicha tarea.

Tabla 4.8: Componente "Centro de Control". Tarea 1

Tarea	
Número de tarea: 1	**Número de componente:** 4
Nombre de la tarea: Diseñar e implementar el módulo “control”.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: El módulo control permite toda la gestión que se hace en el componente Centro de Control, creando el proceso para dicha tarea.	

Tabla 4.9: Componente "Centro de Control". Tarea 2

Tarea	
Número de tarea: 2	**Número de componente:** 4
Nombre de la tarea: Diseñar e implementar el módulo socket.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: El módulo socket permite la conexión de los clientes que se quieran conectar a la Nube.	

Tabla 4.10: Componente "ConexionAPI". Tarea 1

Tarea	
Número de tarea: 1	**Número de componente:** 5

Nombre de la tarea: Diseñar e implementar la librería ConexionAPI.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: La librería ConexionAPI le permite al cliente conectarse a la Nube para poder consumir los servicios requeridos.	

Tabla 4.11: Componente "AplicacionNubeAPI". Tarea 1

Tarea	
Número de tarea: 1	**Número de componente:** 5
Nombre de la tarea: Diseñar e implementar la librería AplicacionNubeAPI.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio:	**Fecha fin:**
Programador responsable: Alejandro Díaz.	
Descripción: La librería AplicacionNubeAPI permite que las aplicaciones puedan desplegar sus servicios en la Nube.	

4.4 Pruebas

4.4.1 Pruebas unitarias

La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es crucial. [7]

4.4.2 Pruebas de aceptación

Las pruebas de aceptación permiten confirmar que la Historia de Usuario (en esta caso seria un componente) ha sido implementada correctamente al final de cada iteración. Este período de prueba

se conoce también como período de caja negra donde se definirán las entradas al sistema y los resultados esperados de estas entradas. Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de éstas es garantizar que los requerimientos han sido cumplidos y que el sistema es aceptable.

A continuación se representan las pruebas de aceptación realizadas para las componentes del sistema:

Tabla 4.12: Caso de Prueba de Aceptación C1_P1 "GestionXML"

Caso de Prueba de Aceptación	
Código: C1_P1.	**Componente:** GestionXML.
Nombre: Verificar funcionamiento de GestionXML.	
Descripción: Prueba para verificar que GestionXML funcione correctamente.	
Condiciones de ejecución: -	
Entrada / Pasos de Ejecución: Se obtiene la expresión binaria de una pieza XML y se le pasa como parámetro a todas las funciones que brinda GestionXML para obtener los resultados esperados.	
Resultado esperado: Todas las funciones que brinda GestionXML dieron los resultados esperados.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.13: Caso de Prueba de Aceptación C2_P1 "Nodo"

Caso de Prueba de Aceptación	
Código: C2_P1.	**Componente:** Nodo.
Nombre: Comunicación del Nodo.	
Descripción: Prueba para verificar el componente Nodo envía y recibe la pieza XML..	
Condiciones de ejecución: -	
Entrada / Pasos de Ejecución: Se le pasa la URL al componente para verificar si la pieza llega al componente Controlador. En el caso de recibir, el Controlador manda al Nodo la expresión binaria de la pieza para verificar que llega a su destino.	

Resultado esperado: La pieza XML llega a su destino sin problemas.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 4.14: Caso de Prueba de Aceptación C3_P1 "Controlador"

Caso de Prueba de Aceptación	
Código: C3_P1.	**Componente:** Controlador.
Nombre: Comunicación del Controlador.	
Descripción: Prueba para verificar que el componente Controlador envía la pieza XML a su destino.	
Condiciones de ejecución: -	
Entrada / Pasos de Ejecución: Se le pasa la expresión binaria de la pieza al componente para verificar si la pieza llega su destino.	
Resultado esperado: La pieza XML llega a su destino sin problemas.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.15: Caso de Prueba de Aceptación C4_P1 "Centro de Control"

Caso de Prueba de Aceptación	
Código: C4_P1.	**Componente:** Centro de Control.
Nombre: Comunicación del Centro de Control.	
Descripción: Prueba para verificar que el componente Centro de Control envía la pieza XML a su destino.	
Condiciones de ejecución: -	
Entrada / Pasos de Ejecución: Se le pasa la expresión binaria de la pieza al componente para verificar si la pieza llega su destino.	
Resultado esperado: La pieza XML llega a su destino sin problemas.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.16: Caso de Prueba de Aceptación C5_P1 "ConexionAPI"

Caso de Prueba de Aceptación	
Código: C4_P1.	**Componente:** ConexionAPI.
Nombre: Comunicación del componente ConexionAPI.	
Descripción: Prueba para verificar que el componente ConexionAPI mantiene la comunicación con la Nube, es decir con el sistema.	
Condiciones de ejecución: El componente Centro de Control tiene que estar funcionando	
Entrada / Pasos de Ejecución: Se ejecuta una aplicación Java sencilla para verificar que se mantenga conectado y envíe ó reciba piezas XML.	
Resultado esperado: La aplicación se mantiene conectada.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.16: Caso de Prueba de Aceptación C5_P1 "AplicacionNubeAPI"

Caso de Prueba de Aceptación	
Código: C4_P1.	**Componente:** AplicacionNubeAPI.
Nombre: Comunicación del componente AplicacionNubeAPI.	
Descripción: Prueba para verificar que el componente AplicacionNubeAPI se comunica con el componente Nodo para que una aplicación pueda desplegar sus servicios.	
Condiciones de ejecución: El componente Nodo tiene que estar funcionando	
Entrada / Pasos de Ejecución: Se ejecuta una aplicación Java sencilla para verificar que se comunica y posibilitando el envío ó recibimiento de piezas XML.	
Resultado esperado: La aplicación se comunica.	
Evaluación de la prueba: Prueba satisfactoria.	

Otra de las pruebas que se va a hacer es desplegar una aplicación que brinde uno o varios servicios determinados para que sean consumidos por clientes.

Aplicación a desplegar:

La aplicación para probar el sistema va a simular una base de datos muy sencilla, donde solo contiene los datos de los productos que gestionan los clientes (nombre del producto, precio, descripción y el cliente que insertó el producto). Primeramente el usuario tiene que estar conectado con la aplicación para después poder insertar productos y consultar los productos existentes. (***Ver el anexo 1 para una mejor observación***)

4.5 Validación

Para validar al sistema se han escogido tres aspectos, *disponibilidad, rendimiento* y *escalabilidad*. A continuación hablaremos sobre esos aspectos.

4.5.1 Disponibilidad

Una empresa que tenga un servidor brindando un servicio determinado, y se requiera de hacer un cambio de código, adicionar un servicio, que cambio el negocio y este la nueva versión lista para desplegarse, el servidor tendrá que ser apagado ó dejar de brindar el servicio, para que el sistema pueda evolucionar. Esta operación incomodaría a los usuarios que se benefician con dichos servicios y además pudiera causar pérdidas a la empresa. Utilizando tecnologías que tengan como característica el reemplazo de código en caliente, el servidor de esta empresa nunca dejaría de brindar los servicios, evitando el descontento de los usuarios y de pérdidas para la empresa, es decir, estaría siempre disponible.

En el lenguaje funcional Erlang la unidad mínima de reemplazo de código es el módulo. Dos versiones distintas del mismo módulo pueden estar cargadas en memoria al mismo tiempo. Cuando se realiza una llamada a una función se utiliza la última versión disponible del módulo.

El reemplazo se puede llevar a cabo llamando a una función requerida escribiendo antes de la función, el módulo en donde se encuentra, seguido por ":".Se muestra un ejemplo de ello en la figura 15.

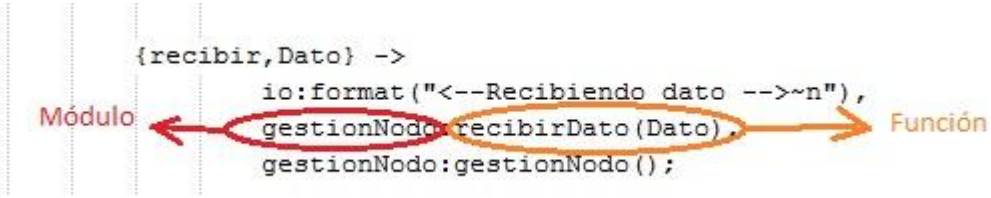


Figura 16: Forma de garantizar el reemplazo de código en caliente.

Para poder validar este punto de referencia, escogimos al componente GestionXML. Primeramente había que tener en cuenta que el sistema estuviera corriendo. El servicio *validad_xml* que brinda este componente no validaba que la pieza XML tuviera los atributos "*from*" y "*to*", los cuales son muy importantes dentro de la pieza. Al hacer una nueva versión del módulo para que pudiera validar lo que anteriormente no validaba, y después de haber compilado el fichero para que estuviera listo para esta operación, los resultados fueron los esperados, ya que el servicio validaba que la pieza XML tuviera los atributos y en ningún momento se apagó el sistema.

4.5.2 Rendimiento

El rendimiento de todo software debe ser el máximo posible, puesto que es beneficioso que el software de una empresa tuviera un tiempo de respuesta rápido, al nivel de los microsegundos. Para el software de comunicación es importante el tiempo en que se demora en llegar el mensaje o los datos que se envió del origen a su destino, y que sea lo mas rápido posible. Esto le da a los usuarios la preferencia del sistema que va a utilizar para comunicarse con los demás. El sistema que le puede aportar mayor porciento de ganancias a una empresa es el de mayor rapidez en las comunicaciones.

Los procesos en Erlang son de peso ligero en el que la máquina virtual no crea un subproceso de sistema operativo para cada proceso creado. Se crean, se programan, y son manipulados en la máquina virtual. Como resultado, el tiempo de creación de un proceso es del orden de microsegundos e independiente del número de procesos existentes simultáneamente. Los procesos de Erlang se comunican entre sí a través del paso de mensajes. A pesar del número de procesos concurrentes en el sistema, el intercambio de mensajes dentro del sistema le toma microsegundos. Un sistema desarrollado en Erlang puede manejar grandes cargas sin degradación en el rendimiento, inclusive sin tener en cuenta la cantidad de procesos existentes.

Para verificar el tiempo que el sistema se demoraba en enviar una pieza XML a su destino se implementó una aplicación sencilla que simulaba la conexión de varios clientes y el envío de miles de

piezas XML. A continuación se presenta una tabla donde se muestra casos donde se envían miles de piezas XML con el tiempo promedio de envío.

Tabla 4.17: Prueba de rendimiento y escalabilidad

Cantidad de Envíos	Tiempo Promedio
30109	0.319
20210	0.295
25030	0.325
35203	0.353
22314	0.280
41020	0.351
33544	0.312
28716	0.342

4.5.2 Escalabilidad

Un sistema que tenga como propiedad la escalabilidad, es muy importante para una empresa, puesto que ese sistema puede crecer manteniendo la calidad de sus servicios y contiene la destreza para operar el incremento de trabajo con fluidez. También puede tener la capacidad de modificar su configuración o tamaño, para ajustarse a los cambios. Podemos citar el ejemplo de una empresa cualquiera que tenga una red de usuarios vía Internet, es lógico que la empresa quiera tener un sistema que le permita trabajar con los clientes actuales, pero también con los posibles clientes, teniendo la oportunidad de cambiar la configuración si así fuese oportuno hacerlo.

El sistema desarrollado en este trabajo de diploma está diseñado arquitectónicamente para que cumpla con los mismos parámetros de escalabilidad que presenta Internet. Nuestro componente Nodo va a estar desplegado en cada entorno de trabajo que contenga una aplicación que brinde servicios en la Nube y permite la comunicación del componente Controlador con la aplicación. El componente Controlador gestiona el envió y recibimiento de piezas XML de todas las aplicaciones dentro de una

red local, a través del componente Nodo, y el componente Centro de Control controla a todos los Controladores que existen en la Nube, es decir en toda la red, además de que mantiene la conexión de los usuarios que consumen los servicios con la Nube. Para que la aplicación pueda desplegar sus servicios en la Nube, debe utilizar la librería AplicacionNubeAPI, la cual permite la comunicación con el componente Nodo. (***Ver los anexos 2 y 3 para una mejor observación***)

Para comprobar que el sistema fuera escalable se ejecutó con la aplicación de prueba que simula la base de datos Productos, corriendo en un entorno de trabajo, y poco a poco se le fueron conectando usuarios para consumir los servicios que brinda la aplicación de prueba. Se estuvo trabajando con 8 usuarios conectados al mismo tiempo y sin problemas de que se conectaran muchos más. Y varios de ellos se desconectaron y se volvieron a conectar sin problemas.

También el reemplazo de código en caliente permite que el sistema sea escalable, ya que con esta característica permite actualizar y agregarle funcionalidades al sistema, sin dejar de prestar los servicios.

4.6 Conclusiones parciales

En este capítulo se desarrolló las etapas de diseño, implementación y prueba del sistema. Para el desarrollo de estas etapas se tomo en cuenta la planificación realiza en el capítulo 3. También se habla sobre los aspectos que se tomaron en cuenta para la validación de dicho sistema.

CONCLUSIONES GENERALES

Para concluir el trabajo desarrollado en este trabajo de diploma, con el fin de desarrollar una infraestructura de comunicación para una plataforma de Cloud Computing, se valora el cumplimiento de las tareas propuestas, donde se propone el estudio de nuevas tecnologías para el desarrollo de sistemas de alta demanda. Con este trabajo se prueba lo ágil y fácil que es desarrollar un software con la tecnología Erlang, en este caso se utiliza el protocolo XMPP como mecanismo de comunicación.

Se obtiene el tipo de middleware Orientado a Mensajes, diseñado e implementado en su totalidad, con un funcionamiento al 100%. También se obtiene el diseño y la implementación del API que utilizarán los clientes para la comunicación con el middleware que va a estar desplegado en la Nube.

Con la obtención de estos dos objetivos, dándole cumplimiento a varias tareas propuestas, se cumplió el propósito de este trabajo de diploma, el cual es la realización de una infraestructura de comunicación para una plataforma de Cloud Computing.

RECOMENDACIONES

Se recomienda realizar un estudio mas profundo con la nueva documentación que ira saliendo en adelante y con la actualización de la existente, para optimizar y hacer más rápida la comunicación del sistema.

BIBLIOGRAFÍA

1. ELSENPETER, ROBERT; VELTE, TOBY J.; VELTE, ANTHONY. *Cloud Computing: A Practical Approach*. Editado por : The McGraw-Hill Companies, 2009. 353 páginas.
2. STANOEVSKA-SLABEVA, KATARINA; WOZNIAK, THOMAS; RISTOL, SANTI. *Grid and Cloud Computing: A Business Perspective on Technology and Applications*. Editado por: Springer, 2009. 273 páginas.
3. LINTHICUM, DAVID. *Cloud Computing and SOA Converge in Your Enterprise*. Editado por: Addison-Wesley, 2009. 265 páginas.
4. CESARINI, FRANCESCO; THOMPSON, SIMON. *Erlang Programing*. Editado por: O´Reilly, 2009. 496 páginas.
5. PUDER, ARNO; ROMER, KAY; PILHOFER, FRANK. *Distributed System Architecture, A Middleware Approach*. Editado por: Morgan Kaufmann Publishers, 2005. 342 páginas.
6. GRAHAM, IAN. *Requirements Modeling and Specification for Service Oriented Arquitecture*.
7. Beck, Kent. *Extreme Programming Explained*. Editado por: John Wiley & Sons, 2000. 318 páginas.
8. LETELIER, PATRICIO. *Metodologías ágiles para el desarrollo de software*. 2004.
9. REESE, GEORGE. *Cloud Application Architectures, Building Applications and Infrastructure in the Cloud*. Editado por: O´Reilly, 2009. 206 páginas.
10. RITTINGHOUSE, JOHN; RANSOME, JAMES. *Cloud Computing: Implementation, Management and Security*. Editado por: CRC Press, 2009. 340 páginas.
11. VAN STEEN, MAARTEN; TANENBAUND, ANDREW. *Distributed Systems: Principles and Paradigms*. Editado por: Upper Saddle River, 2006. 705 páginas.
12. KSHEMKALYANI, AJAY; SINGHAL, MUKESH. *Distributed Computing: Principles, Algorithms and Systems*. Editado por: Cambridge, 2008. 756 páginas.
13. MAHMOUD, QUSAY. *Middleware for Communications*. Editado por: John Wiley & Sons, 2004. 524 páginas.
14. SAINT-ANDRE, PETER; SMITH, KEVIN; TRONCON, REMKO. *XMPP: The Definitive Guide*. Editado por: O´Reilly, 2009. 306 páginas.
15. COBAS MORA, YAIMA y ALMAGUER GUERRERO, AYLIN. *Implementación Distribuida del Gaussian sobre la Plataforma JDCS*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana.

16. SOSA CEDEÑO, IRINA y DUHARTE PEÑA, ERNESTO. *Sistema para Distribuir los Cálculos de la Aplicación Vega sobre la plataforma Java Distributed Computing System (JDCS)*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana.
17. GER PEÑA, RAICEL. *Sistema de Comunicación (Middleware) aplicable a diferentes entornos distribuidos*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana.
18. ABREU MACEO, RAYMOND y ARAVELO PÉREZ, LISANDRA. *Propuesta de un Lenguaje de Descripción para SOA*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana.
19. Slideshare. Rational Rose. 2008 [cited 28/01/09]; Available from: http://www.slideshare.net/vivi_jocadi/rational-rose.

REFERENCIAS BIBLIOGRÁFICAS:

1. COBAS MORA, YAIMA y ALMAGUER GUERRERO, AYLIN. *Implementación Distribuida del Gaussian sobre la Plataforma JDCS*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana. Capitulo 1, páginas 4-10.
2. SOSA CEDEÑO, IRINA y DUHARTE PEÑA, ERNESTO. *Sistema para Distribuir los Cálculos de la Aplicación Vega sobre la plataforma Java Distributed Computing System (JDCS)*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana. Capitulo 1, páginas 5.
3. GER PEÑA, RAICEL. *Sistema de Comunicación (Middleware) aplicable a diferentes entornos distribuidos*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana. Capitulo 1, páginas 16-27.
4. ABREU MACEO, RAYMOND y ARAVELO PÉREZ, LISANDRA. *Propuesta de un Lenguaje de Descripción para SOA*. Trabajo de Diploma para optar por el título de ingeniero informático. Universidad de las Ciencias Informáticas 2008. C. Habana. Capitulo 1, páginas 7 y 8.
5. Slideshare. Rational Rose. 2008 [cited 28/01/09]; Available from: http://www.slideshare.net/vivi_jocadi/rational-rose.
6. Letelier, Patricio. *Metodologías ágiles para el desarrollo de software*.2004.
7. Beck, Kent. *Extreme Programming Explained*.2000.

GLOSARIO DE TÉRMINOS Y SIGLAS

Concurrencia: concepto que define la existencia de muchas interacciones simultaneas con un objeto (para el caso de los servidores, un servidor que permita concurrencia, es un servidor que puede manejar múltiples conexiones de clientes sin afectar el servicio para ninguno, donde los clientes no tienen que esperar en cola por la respuesta).

Erlang/OTP: Entorno de desarrollo para la programación concurrente de Erlang.

Erlang: Lenguaje de programación funcional de alto nivel para la el diseño de sistemas en tiempo real.

Función: Los programas Erlang son escritos enteramente en términos de módulos con funciones. Una función puede tener argumentos y siempre retorna un valor. Las funciones pueden ser exportadas y estar así disponibles para llamarlas desde otros módulos. Las funciones no exportadas solo pueden usarse dentro del módulo al que pertenecen.

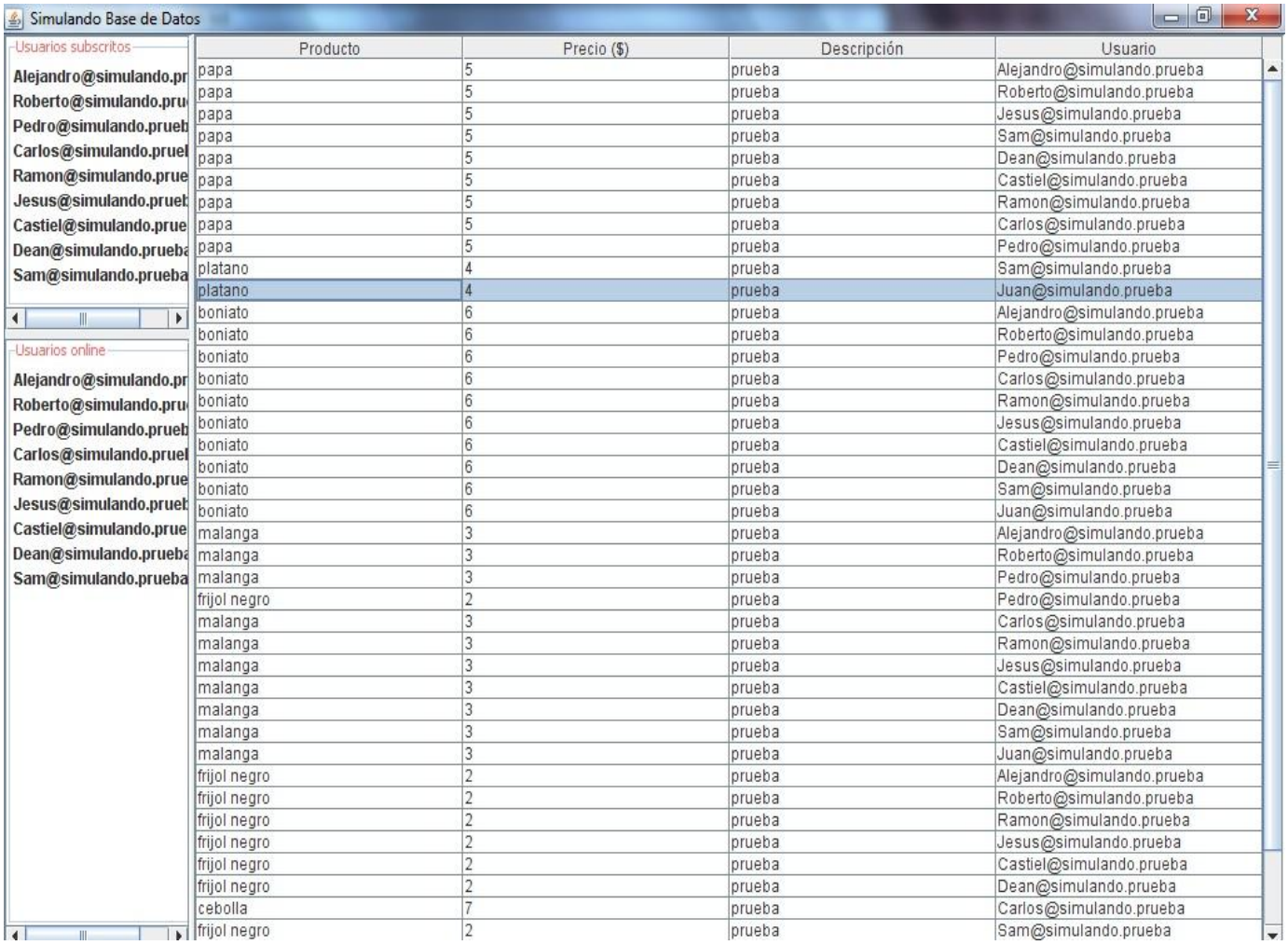
Máquina Virtual de Erlang: Hace posible que Erlang/OTP trabaje en cualquier tipo de sistema operativo o plataforma. La Máquina Virtual de Erlang está disponible en diferentes plataformas.

Módulo: Un módulo es una unidad para la compilación y carga en Erlang. Un módulo contiene declaraciones y código representando a funciones dentro del módulo.

Nodo: Una instancia de la Máquina Virtual de Erlang, que se puede comunicar con otras instancias de la misma (otros nodos) dentro del entorno Erlang/OTP.

OTP: Plataforma Abierta de Telecomunicación.

ANEXO 1



Producto	Precio ($)	Descripción	Usuario
papa	5	prueba	Alejandro@simulando.prueba
papa	5	prueba	Roberto@simulando.prueba
papa	5	prueba	Jesus@simulando.prueba
papa	5	prueba	Sam@simulando.prueba
papa	5	prueba	Dean@simulando.prueba
papa	5	prueba	Castiel@simulando.prueba
papa	5	prueba	Ramon@simulando.prueba
papa	5	prueba	Carlos@simulando.prueba
papa	5	prueba	Pedro@simulando.prueba
platano	4	prueba	Sam@simulando.prueba
platano	4	prueba	Juan@simulando.prueba
boniato	6	prueba	Alejandro@simulando.prueba
boniato	6	prueba	Roberto@simulando.prueba
boniato	6	prueba	Pedro@simulando.prueba
boniato	6	prueba	Carlos@simulando.prueba
boniato	6	prueba	Ramon@simulando.prueba
boniato	6	prueba	Jesus@simulando.prueba
boniato	6	prueba	Castiel@simulando.prueba
boniato	6	prueba	Dean@simulando.prueba
boniato	6	prueba	Sam@simulando.prueba
boniato	6	prueba	Juan@simulando.prueba
malanga	3	prueba	Alejandro@simulando.prueba
malanga	3	prueba	Roberto@simulando.prueba
malanga	3	prueba	Pedro@simulando.prueba
frijol negro	2	prueba	Pedro@simulando.prueba
malanga	3	prueba	Carlos@simulando.prueba
malanga	3	prueba	Ramon@simulando.prueba
malanga	3	prueba	Jesus@simulando.prueba
malanga	3	prueba	Castiel@simulando.prueba
malanga	3	prueba	Dean@simulando.prueba
malanga	3	prueba	Sam@simulando.prueba
malanga	3	prueba	Juan@simulando.prueba
frijol negro	2	prueba	Alejandro@simulando.prueba
frijol negro	2	prueba	Roberto@simulando.prueba
frijol negro	2	prueba	Ramon@simulando.prueba
frijol negro	2	prueba	Jesus@simulando.prueba
frijol negro	2	prueba	Castiel@simulando.prueba
frijol negro	2	prueba	Dean@simulando.prueba
cebolla	7	prueba	Carlos@simulando.prueba
frijol negro	2	prueba	Sam@simulando.prueba

Figura 17: Interfaz de usuario de la aplicación de prueba desplegada con varios usuarios en línea y con productos insertados.

ANEXO 2

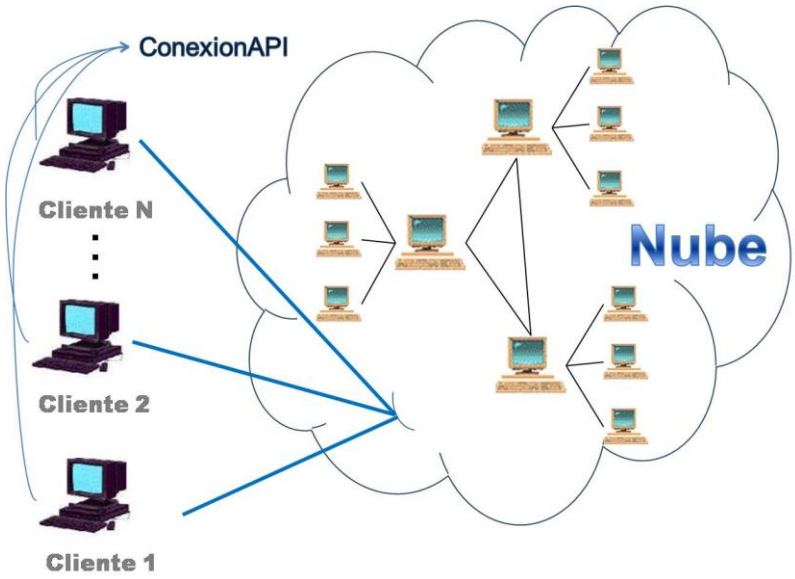


Figura 18: Ilustración de cómo los clientes se conectan con la Nube, utilizando la librería ConexionAPI.

ANEXO 3

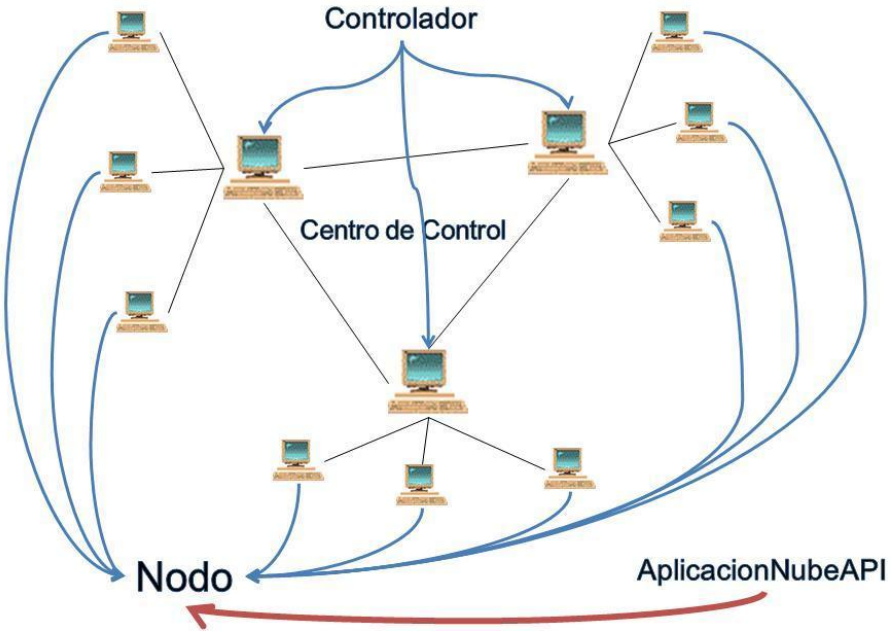


Figura 19: Ilustración de cómo se despliega cada componente en la red. El componente Centro de Control va ubicado en cualquier entorno de trabajo o servidor.