Proyecto de Grado

Desarrollo de un Servicio Web para el Simulador de Eventos Discretos GALATEA

Por

Br. Gustavo J. Marcano V.

Tutor: Dr. Jacinto A. Dávila Q.

Marzo 2015

Desarrollo de un Servicio Web para el Simulador de Eventos Discretos GALATEA

Br. Gustavo J. Marcano V.

Proyecto de Grado — Sistemas Computacionales, 64 páginas

Resumen: En éste trabajo, se acomete implementar y desplegar como servicio, una aplicación utilizada para modelar y simular eventos discretos en sistemas distribuidos, utilizando como medio los servicios web. Esto es conocido como Simulación Distribuida, la cual posee una serie de protocolos para su desarrollo e interoperabilidad. Dicha interoperabilidad se consigue mediante la adopción de estándares abiertos, como es el caso de XML, que habrá de fungir como el medio encargado de transportar los mensajes en un formato estandarizado lo cual lo hace accesible a prácticamente cualquier dispositivo. Como su nombre lo propone, en éste caso se hace uso del paradigma de arquitectura orientada a servicios (SOA: Service Oriented Architecture).

Palabras clave: Servicios Web, Sistemas Distribuidos, Web Semántica, Arquitectura Orientada a Servicios

Índice

Índice de Tablas vi

Índice de Figuras vii

1 Introducción 1

1.1 Antecedentes . . . 1
1.2 Planteamiento del Problema . . . 2
1.3 Objetivos . . . 3
1.3.1 Objetivos Generales. . . . 3
1.3.2 Objetivos Específicos. . . . 3
1.4 Justificación. . . . 3
1.5 Metodología. . . . 4
1.5.1 Desarrollo en Espiral. . . . 4

2 Marco Teórico 10

2.1 Servicio Web . . . 10
2.1.1 Basado en XML . . . 11
2.1.2 Débilmente Acoplado . . . 11
2.1.3 Desarrollo de refinado basto . . . 11
2.1.4 Capacidad de ser síncrono y asíncrono . . . 12
2.1.5 Soporte de llamada a procedimientos remotos . . . 12
2.1.6 Soporte de intercambio de documentos . . . 12
2.2 SOAP . . . 14
2.2.1 Esquema de SOAP . . . 15

2.2.2 Componentes de SOAP . . . 15
2.2.3 WSDL . . . 17
2.3 REST . . . 17
2.4 Arquitectura orientada a servicios . . . 18
2.5 OpenAM . . . 19
2.5.1 Servicios OpenAM . . . 20
2.5.2 Federación . . . 21
2.5.3 SAML 2.0 . . . 22
2.5.4 Perfil SSO del navegador web . . . 22
2.6 Simulación Distribuida . . . 23
2.7 Simulación de Eventos Discretos . . . 23
2.7.1 Descripción de GALATEA . . . 24
2.7.2 Diseño de GALATEA . . . 25

3 Modelado y análisis de requerimientos 27
3.1 Iteración 1 . . . 27
3.1.1 Determinar Objetivos. . . . 28
3.1.2 Análisis del riesgo. . . . 28
3.1.3 Desarrollar y probar. . . . 29
3.1.4 Planificación . . . 29
3.2 Iteración 2 . . . 30
3.2.1 Determinar los objetivos . . . 30
3.2.2 Análisis del riesgo. . . . 30
3.2.3 Desarrollar y probar. . . . 30
3.2.4 Planificación . . . 31
3.3 Iteración 3 . . . 32
3.3.1 Determinar los objetivos . . . 33
3.3.2 Análisis de riesgo . . . 33
3.3.3 Desarrollar y probar. . . . 34

4 Diseño, implementación y prueba del Sistema 37
4.1 Selección del sistema a utilizar . . . 37

4.1.1 Gestión del servicio y usuario 37
4.1.2 Especificación del sistema 38
4.2 Arquitectura de la interfaz de usuario 40

5 Conclusión **43**
5.1 Apéndices 45
5.1.1 Instalación de Java 45
5.1.2 Instalación de eclipse 47
5.1.3 Instalación de Apache Tomcat 49
5.1.4 Instalación de Axis 2 51
5.1.5 Instalación de OpenAM 52
5.1.6 Segmento de código de SAML 55

Bibliografía **62**

Índice de Tablas

Índice de Figuras

1.1 Esquema del modelo en espiral, tomado de (Icke, 2007) 5

2.1 Diagrama de Componente SSO 13
2.2 Diagrama de Componente Cliente-Servidor SOA. 13
2.3 Arquitectura de un servicio web basado en SOAP. Tomado de (Kalin, 2012). 15
2.4 Ilustración de un sobre SOAP. Tomado de (Sokpop, 2009) 16
2.5 Tomado de (Kalin, 2012) 18
2.6 (Kalin, 2012) 18
2.7 Arquitectura de OpenAM. (Craig, 2013) 20
2.8 Diagrama de secuencia SAML. Tomado de (Trscavo, 2007) 22
2.9 Glider. Motor simulador y clases básicas GALATEA 26

3.1 Demo de un servicio web. 29
3.2 Panel de Administrador OpenAM. 31
3.3 Creando fedlet OpenAM. 32
3.4 Interfaz demo de fedlet OpenAM. 32
3.5 Prueba exitosa de fedlet. 33
3.6 Estadísticas de Galatea. 34
3.7 Trazas de corrida de Galatea. 35
3.8 Interfaz De Usuario Galatea. 35
3.9 Registro De Usuario Galatea. 36
3.10 Registro invalido Usuario Galatea. 36

4.1 Diagrama mas detallado de interacción 40

4.2 Arquitectura interfaz de usuario 42

5.1 Servidor Apache Tomcat 51
5.2 Servidor Axis2 iniciado 52
5.3 Modificando el script. 54
5.4 Interfaz de usuario. 56

Capítulo 1

Introducción

El siguiente estrato, introduce una descripción de como se ha desarrollado el servicio que prestará una aplicación basada en simulación de eventos discretos (DEVS), que estará fundamentado en GALATEA (Uzcátegui, 2011), el cual es un software para simulación de sistemas multi-agentes producto de dos líneas de investigación: lenguajes de simulación basados en la teoría de simulación de Zeigler y agentes basados en lógica. Esta concepción, está enfocada en los nuevos paradigmas de la arquitectura orientada a servicios (SOA), por el que, a partir funciones débilmente acopladas proveerá la capacidad de intercambiar funcionalidades de componentes dando escalabilidad a futuras extensiones. El propósito es que sirva para proveer un servicio de simulación de evento discretos (DEVS) al usuario desde cualquier punto. Así también, se pretende analizar conceptos de seguridad mediante el cual, ante una petición al computador (request), ésta sea procesada solo si ha sido previamente validada. Dicho mecanismo es denominado SAML (Security Assertion Markup Language) (Fawcett, 2012).

1.1 Antecedentes

Como trabajos similares a esta propuesta se puede mencionar los realizado por (Uzcátegui, 2011) el cual relata la fase de desarrollo, cómo surge y hacia donde se proyecta GALATEA. Implementada como una plataforma libre de código abierto para simulación de sistemas multi-agente que incorpora estrategias de simulación bien

conocidas con la que cualquier modelista o simulista puede ensayar dichas estrategias en problemas de simulación de sistemas complejos. Por otra parte, otro precedente tomado en cuenta ha sido (Rengifo, 2011), el cual trata de una tesis de pre-grado que expone el desarrollo de un servicio web para la Modeloteca del Sistema Nacional de Simulación. La misma consistió en buscar, mediante el uso de una aplicación web, la solución al problema existente en el Centro de Simulación y Modelado (CESIMO) de la Universidad de Los Andes, en el cual habían dificultades para mantener un registro referente a los proyectos que allí se desarrollaban, lo que como resultado provocaba que a menudo se perdiera información referente a los mismos, ó que incluso se llevara a cabo proyectos de forma innecesaria, ya que trabajos parecidos habrían sido realizado antes. Como consecuencia, debido a la falta de un repositorio institucional, resultaban imposibles de conseguir. A partir de estos dos antecedentes lo que se pretende es, tomando como base GALATEA, adaptarlo como un servicio web en el cual el usuario pueda interactuar con el simulador, sin tener las limitaciones como por ejemplo suelen surgir al usar sistemas operativos o navegadores diferentes, incluso ante arquitecturas distintas. A su vez, de (Rengifo, 2011), se busca obtener la experiencia ganada al desarrollar un Servicio Web en sí.

1.2 Planteamiento del Problema

En la medida que se ha avanzado en términos tecnológicos, se percibe un crecimiento proporcional de la información que se genera en todos los ámbitos, sea científico, humanístico, económico etc. Estos fenómenos no suceden como acontecimientos aislados, sino que son posibles gracias al conocimiento adquirido y al esfuerzo de muchos, de ponerlo a disposición de quien desee acceder a ellos. Teniendo en cuenta esta premisa, surge el planteamiento del problema, el cual tiene dos vertientes, la primera, consiste en que no se dispone de una plataforma acondicionada para que, de manera fácil y rápida se pueda hacer uso del simulador GALATEA. Si bien se posee un servidor académico en CESIMO, es necesario desarrollar mecanismos que posibiliten el uso de las herramientas que éste ofrece, por ello la necesidad de servicios web que permitan que las funcionalidades que posee el simulador estén disponibles tanto a nivel

local como a nivel externo (Sistema Distribuido). En cuanto al otro aspecto, consiste en verificar que la integridad de la información estén garantizados, en especial cuando el usuario desee enviar algún dato, o realizar alguna consulta, por lo cual el sistema debe validar a dicho usuario previamente.

1.3 Objetivos

1.3.1 Objetivos Generales.

Desarrollar una aplicación web para un simulador de eventos discretos basado en GALATEA, implantándole una interfaz gráfica de usuario (GUI) y establecerla en un servicio web funcional para su posterior corrida y estudio.

1.3.2 Objetivos Específicos.

- Establecer un servicio web funcional mediante el uso de herramientas y técnicas enfocadas a tal fin como es el caso SOA (Service Oriented Architecture).
- Dominar el funcionamiento que proporciona los proveedores de servicios que se encargarán de gestionar la base de datos de usuarios. Se evaluarán las plataformas virtuoso u OpenAM para este propósito.
- Incorporar el simulador GALATEA como servicio web.
- Diseñar y desarrollar un cliente GUI/controlador para un modelo que se pueda simular con GALATEA a través del servicio web implementado.
- Sistematizar la experiencia de uso del servicio web para simulación.

1.4 Justificación.

La principal motivación para realizar este trabajo, es la de desarrollar un servicio web que sirva como base para la simulación de eventos discretos de sistemas que utilizan la tecnología de agentes con el fin de desplegarlo para redes externas (Sistema Distribuido)

todo ello a partir del servidor académico localizado en CESIMO, cuyo nombre se ha denominado “CHES”.

1.5 Metodología.

En cuanto a la metodología usada para la implementación del software, se visualiza desde dos ópticas fundamentales. La primera, referente al marco de trabajo a usar, que de acuerdo a las características del proyecto se adecúa mejor al “Ciclo de vida del Desarrollo de software” (Boehm, 1986). Esto se justifica, dada cierta complejidad que proporciona el proceso, así como también la capacidad de obtener modelos funcionales en periodos de tiempos relativamente cortos, además de que a lo largo de su existencia, ha demostrado su gran utilidad. En lo referente al segundo enfoque se trata del conjunto de actividades que conducirán a la creación del producto. Estos son llamados modelos de procesos. Para el caso particular de este proyecto, se procede con el modelo en espiral, el cual corresponde a una sub-sección del llamado proceso iterativo.

1.5.1 Desarrollo en Espiral.

El modelo en espiral fue originalmente propuesto por (Boehm, 2000). Mas que representar el proceso de software como una secuencia de actividades con retrospectiva de una actividad a otra, se presenta como una espiral. Cada ciclo en la espiral representa una fase del proceso de software. Así, el ciclo mas interno podría referirse a la viabilidad del sistema, el siguiente ciclo a la definición de requerimientos, el siguiente ciclo al diseño del sistema, y así sucesivamente, la figura 1.1, lo ilustra de mejor manera.

Cada ciclo de la espiral se divide en cuatro sectores:

1. Definición de objetivos.
2. Evaluación y reducción de riesgos.
3. Desarrollo y validación.
4. Planificación.

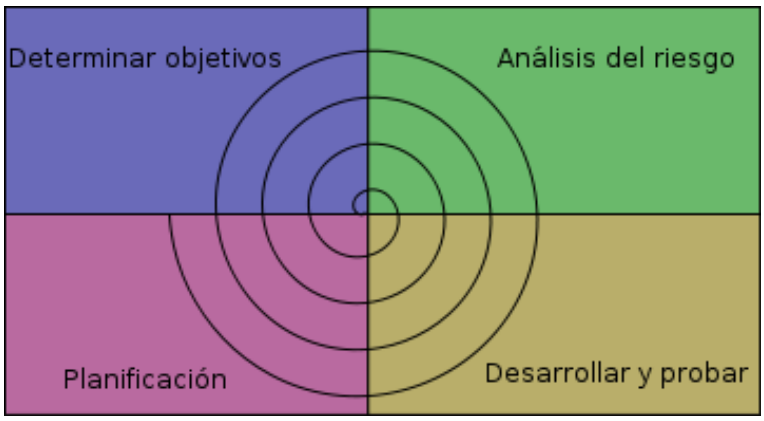

Figura 1.1: Esquema del modelo en espiral, tomado de (Icke, 2007)

En las fases iniciales para el desarrollo, se discutió acerca de las posibles tecnologías y herramientas necesarias para la elaboración del proyecto, que posteriormente demostraron en su mayoría que cumplían con la apreciación inicial.
Entre las tecnologías evaluadas y que fueron necesarias para la consecución del objetivo se listan las siguientes:

- GNU Linux "Ubuntu 12.04.5 LTS precise x86_64". Esto justificado por varias razones, la fundamental es que se trata de software de código abierto, pero además por la robustez del mismo, junto con la facilidad de documentación para la mayoría de tecnologías utilizadas.

- Entorno de desarrollo integrado "Eclipse Java EE IDE for Web Developers"; que consiste en un editor enriquecido para programar con lenguaje Java.

- Lenguaje de programación Java como base, principalmente para lo que constituye la lógica y funcionalidades del programa en sí. Específicamente se hace uso de la version "jdk1.7.0_45" que constituye el kit de desarrollo java para programadores.

  - GALATEA como herramienta base para la corrida de modelos de simulación a través de un servicio web.

  - JSP (Java Server Pages) como parte de la interfaz de usuario mediante la que se podrá interactuar con el sistema.

  - OpenAM la cual es una herramienta con un conjunto de API para el desarrollo de Fedlets (Federaciones) que servirán para la manejo de autenticación y autorización en el uso de los servicios de aplicaciones web, esto enmarcado en lo que es SAML (Security Assertion Markup Language).
  - JAX-RS (Java API for RESTful Web Services); el cual es un API para la elaboración de servicios web con el estilo arquitectónico REST, (Representational State Transfer).
  - JAX-WS (Java API for XML Web Services; el cual consiste también en un API de java para la creación de servicios web, pero a diferencia del anterior, JAX-WS está enfocado en SOAP, el cual es un paradigma distinto al REST. Esto será explicado de manera mas detallada en el próximo capitulo.

- Virtuoso; el cual es un servidor universal con motor de gestor de base de dato relacional (RDBMS) y servidor de aplicación web todo integrado en una sola herramienta. Dicha herramienta estuvo en la evaluación de riesgo preliminar, la cual consistió en la puesta en marcha de instalación, y pruebas de corridas de elementos que lo conforman. Dada la complejidad y falta de entendimiento de la puesta en marcha de dicha herramienta, se desistió del mismo. Todo esto se hizo como parte de la metodología en espiral, y si bien posteriormente se optó por no usarla, es menester comentar la experiencia.

- Servidor Web Apache 2.2, que funciona como servidor HTTP (Hypertext Transfer Protocol) y que a su vez es de código abierto.

- Servidor de aplicaciones web Apache tomcat, que sirve contenedor de servlets y de JSP.

- Servidor para servicios web Apache Axis2; que fungirá como el motor de servicios web y que se complementa muy bien con las herramientas antes mencionadas pues forman parte de la fundación Apache para desarrollo de este tipo de herramientas.

1.5.1.1 Definición de objetivos.

- Establecer un sistema funcional de GALATEA en el cual varios de los modelos disponibles puedan ser corridos a través de la implementación de Servicios Web.
- Gestionar el mecanismo para autenticación y autorización de usuarios, los cuales permitirán disponer de los servicios de GALATEA, (dichos mecanismos ya existen y han sido implementado por terceras partes). Estos consisten en un estándar abierto basado en XML que tiene por nombre Lenguaje de Marcado para Confirmaciones de Seguridad (SAML por sus siglas en inglés). SAML está constituido en tres roles:
  - Principal
  - Proveedor de identidad
  - Proveedor de servicio
- Integrar los distintos elementos que componen el sistema en su conjunto. Esto se logra mediante el paradigma de la arquitectura orientada a servicio (SOA), cuya filosofía basada en desarrollar componentes débilmente acoplados posibilita la unificación de los mismos.

1.5.1.2 Riesgos

- El tener que aprender como funciona, y como se ha debe manejar software desarrollados por terceros, así como también la integración que se vaya a establecer entre los distintos componentes del sistema.
- Establecer en qué áreas se trabajará, es decir, tomando en cuenta que GALATEA fue desarrollado entre varios colaboradores, se hace necesario determinar qué parte va a tener éste proyecto dentro de la visión mas amplia del sistema.
- Incompatibilidad entre los componentes que conformarán el sistema, ya sea debido a versiones diferentes, formatos utilizados, o inclusive las plataformas sobre las que fueron desarrollados, etc.

- Posibilidad de falta de sincronía en las fases de desarrollo debido a la separación geográfica de la mayoría de integrantes que intervienen en el sistema.

- Irrupción de propiedades emergentes que conlleva la implantación de cualquier sistema. Es decir, la posibilidad de que surjan inconvenientes no previstos, que aún habiendo sido evaluadas en fase de diseño y modelado, pueden no ser perceptibles.

1.5.1.3 Evaluación y reducción de riesgos.

- Al hacer análisis de requerimientos, se buscaron en muchos casos al menos una o dos alternativas de tecnologías que cubrieran los requisitos. Algunos ejemplos se pueden hallar en los manejadores de base de datos (MySQL, PostgreSQL), o la del manejo de proveedor de servicios y/o de identidades (Virtuoso, OpenAM), todos estas medidas han sido tomadas para poder contar con mas de una opción al momento de la implementación .

- En lo referente al riesgo de incompatibilidad, una vez evaluado el medio, se concluye que la mejor manera (y lógica) de reducirlo es haciendo revisión de la documentación del software previamente desarrollado, en este caso GALATEA, y además de ello, establecer contactos con los desarrolladores y creadores del mismo. (Rengifo, 2011)

- Se evaluó qué herramientas serían mas apropiada para el manejo de las identidades. Después de analizar, se determinó la existencia de dos que cubren los criterios para emprender el desarrollo (que sean de código abierto). La primera de dichas herramientas trata de Virtuoso, y la segunda, OpenAM. Ambas ofrecen servicios que son críticos en la propuesta a desarrollar, sobre todo en cuanto al manejo de base de datos de usuarios, ya que favorece el rápido avance, al no tener que diseñar modelos de datos desde el inicio.

- Otras de las dificultades a reducir es la poca experiencia en implementar “Servicios Web”, que en consecuencia de ello, cuenta con el apoyo y experticia, principalmente del tutor, y también de el grupo de trabajo.

- A través del uso de las variadas herramientas de comunicación disponibles (correos, repositorios, vídeo-conferencias, almacenamiento en nube, etc.) se busca reducir las limitaciones o dificultades de sincronía que pudiera generarse debido a la separación geográfica.

1.5.1.4 Desarrollo y validación.

Después de la evaluación de riesgos se elige el modelo para el desarrollo del sistema. Dadas las características del sistema en cuestión se extrae un par de técnicas que recogen las mejores practicas para la obtención del producto. Consecuentemente con lo que se ha planteado hasta hora, una consiste en el modelo en espiral. Adicionalmente a ello, se extrae el modelo basado en componente (Component-based software engineering, CBSE). La segunda está sustentada precisamente en el tipo de proyecto que se desarrolla. Tomando en cuenta que la filosofía de esta técnica, consiste en la reutilización y separación en elementos de software mas pequeños para el fácil acoplamiento de las partes del sistema, respalda la elección.

Capítulo 2

Marco Teórico

En éste capítulo se explica los términos, conceptos y todo el conjunto de tecnologías que serán necesarias para la consecución del proyecto que se propone. Por una parte desglosar qué son servicios web, su uso, componentes y ventajas. En el otro lado está la simulación de eventos discretos DEVS (Discrete Event Simulation), cuya utilidad en ambientes académicos está bien establecida. Ambas deberán integrarse para poder ser desplegado a través de sistemas distribuidos.Aunque pudiera parecer una operación trivial, se trata de un trabajo que supone dificultades a tener en cuenta. Se define detalladamente la arquitectura orientada a servicios SOA (Service Oriented Architecture), la cual es la que proporciona las pautas y estructuras de como establecer un servicio web. En cuanto al desarrollo del simulador de eventos discretos, como se mencionó anteriormente la referencia es GALATEA, con el fin de proporcionar una base inicial para la codificación del nuevo sistema.

2.1 Servicio Web

Un servicio web es visto como una pieza de lógica de negocios que se encuentra en la Internet, y que es accesible a través de protocolos estándar de dicha Red, tales como HTTP, SMTP. El uso de un servicio web puede ser tan simple como una simple como acceder a un sitio (logging) ó tan complejo como facilitar una negociación a una multiorganización (Kalin, 2012).

Dada esta definición, muchas tecnologías usadas reciente y no tan recientemente, pudieran ser clasificadas como servicios web pero no lo son. Estas incluyen: “win32 technologies”, J2EE, CORBA, and “CGI scripting”. La gran diferencia entre estas tecnologías y las nuevas que son catalogadas como servicio web, es la estandarización. Estas nuevas clases están basadas en XML estandarizado (a diferencia del propietario estándar binario) y es soportado globalmente por la mayoría de las mas grandes firmas de tecnología. XML provee una forma de lenguaje neutral para representar datos, y el soporte corporativo y de organizaciones garantizará que todo software importante tendrá una estrategia de servicios web para los próximos años.
Un Servicio web tiene características de comportamientos especiales:

2.1.1 Basado en XML

Con el uso de XML como el protocolo de servicios web para todas las capas de representación de datos que son creados, estas tecnologías serán interoperable a nivel nuclear (core level). Como transporte de datos, XML elimina cualquier atadura a red, sistema operativo o plataforma que el protocolo tenga.

2.1.2 Débilmente Acoplado

El consumidor de un servicio web, no está ligado directamente a un servicio web; la interfaz puede cambiar con el tiempo sin comprometer la habilidad del cliente de interactuar con dicho servicio. Un sistema fuertemente acoplado implica que el cliente y la lógica del servidor están estrechamente relacionado el uno con el otro, lo que se supone que si uno cambia, el otro debe ser actualizado. Adoptando una arquitectura débilmente acoplada tiende a hacer los sistemas de software mas manejable y permite una una integración mas simple entre distintos sistemas.

2.1.3 Desarrollo de refinado basto

Las tecnologías orientada a objetos como tales como Java, exponen sus servicios a través de métodos individuales. Un solo método constituye una operación “muy fina” para proveer alguna habilidad a niveles mayores. El construir un programa en el lenguaje

Java desde cero, requiere la creación de múltiples métodos altamente refinados que son luego compuestos en un servicio bastamente refinado (abstracción de datos) que es consumido ya sea por un cliente o por otro servicio. Los negocios y las interfaces que se despliegan deben de ser bastamente refinado. Las tecnologías de servicio web proveen una forma natural de definir servicios bastamente refinados que acceden a la cantidad correcta de lógica de negocios.

2.1.4 Capacidad de ser síncrono y asíncrono

Síncrono se refiere a que el cliente está atado a la ejecución del servicio. En las invocaciones síncronas, el cliente bloquea (el proceso) y espera por el servicio para completar su operación antes de continuar. Las operaciones Asíncrona permite al cliente invocar un servicios y luego ejecutar otras funciones. Estos, pueden obtener resultados en un momento posterior mientras que los clientes síncronos reciben los resultados cuando el servicio los haya completado. La capacidad asíncrona es un factor esencial para habilitar los sistemas débilmente acoplados.

2.1.5 Soporte de llamada a procedimientos remotos

Los servicios web permite a los clientes invocar procedimientos, funciones y métodos en objetos remotos usando un protocolo basado en XML. Los procedimientos remotos exponen parámetros de entradas y salidas que un servicio web debe soportar. El desarrollo en componentes vía EJB (Enterprise Java Beans) y .NET, se han convertido gradualmente en parte de las arquitectura y de los despliegues por parte de empresas por varios años. Ambas tecnologías son distribuidas y accesibles a través de una variedad de mecanismos RPC. Un servicio web soporta RPC al proveer servicios propios, equivalente a un componente tradicional, o "traduciendo" invocaciones entrantes a una invocación EJB o un componente .NET.

2.1.6 Soporte de intercambio de documentos

Una de las ventajas claves de XML es su forma genérica de no solo representar datos, pero también documentos complejos. Estos documentos pueden ser simple, tales como

una dirección actual, o tan complejo como representar un libro completo o RFQ. Los servicios web soportan el intercambio transparente de documentos para facilitar la integración de negocios.

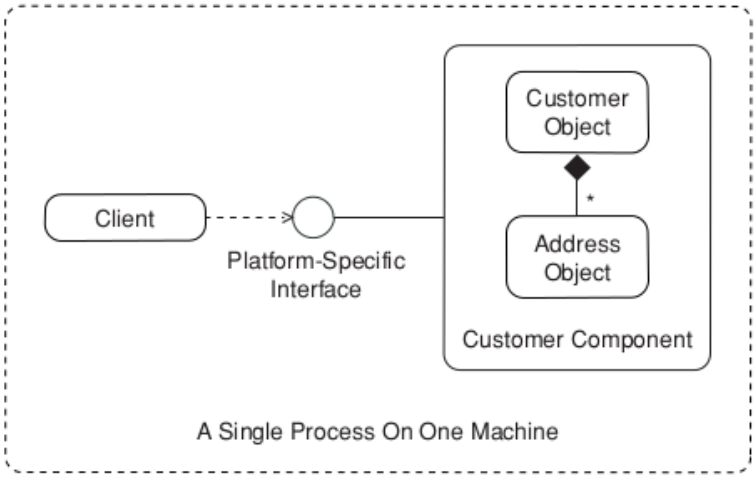

Figura 2.1: Diagrama de Componente SSO

Los componentes fueron vistos como un medio para facilitar el reuso a través de lenguajes de programación dispares. Desafortunadamente, eran creado para plataformas computacionales especificas.Tomado de (Daigneau, 2012).

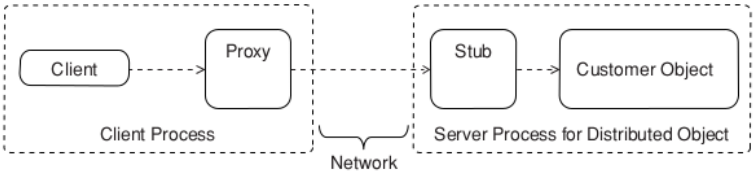

Figura 2.2: Diagrama de Componente Cliente-Servidor SOA.

Los objetos fueron usados frecuentemente en escenarios distribuidos. Cuando un cliente invocaba un método en la interfaz del proxy, el proxy despachaba la llamada por la red a un stub remoto, y el método correspondiente al objeto distribuido sería invocado. Mientras que el cliente y el objeto distribuido usaran la misma tecnología, todo funcionaba bien. Tomado de (Daigneau, 2012)

Mientras que el servicio web inicialmente estaba destinado a ser un protocolo de acceso a objetos simple (SOAP por sus siglas en ingles) para acceder a métodos de

objetos, tanto local como remotamente, el propósito ha cambiado significativamente desde entonces. Actualmente,la perspectiva común es que los servicios web provean funcionalidad, independientemente de la tecnología de implementación y del lugar donde se ejecute. La noción conceptual de objeto ya no juega un rol. En cambio, el concepto de "interfaz" tomó su lugar. Esto se puede observar en elementos, como la identificación de objetos únicos, los métodos, clases, instancias, herencias y otros componentes, no forman parte del modelo de los servicios web en absoluto. Los servicios web proporcionan la noción de una interfaz independiente del lenguaje de programación utilizado para implementarla. Existen varias interpretaciones conceptuales de los servicios web que se han discutido durante bastante tiempo y siguen siendo objeto de debate. Estas interpretaciones se manifiestan en varios "estilos" como REST, XML-RPC, SOAP, y documento, por citar los más importantes. El estilo REST sigue los principios de la comunicación a la manera de URL, el XML-RPC sigue la idea de invocación remota, SOAP aboga por la abstracción del protocolo de transporte, y el estilo documento plantea la noción de documento en lugar de datos de objetos (Fensel, 2008). Schema es un término que usan los ingenieros de tecnologías de información (IT) que describe la estructura y organización de un conjunto de datos - por lo general los datos contenidos en una base de datos. Así, XSD es un lenguaje para describir exhaustivamente la estructura y organización de los datos - . Se puede utilizar para describir cualquier tipo de datos: números, fechas, textos, listas, registros, bases de datos enteras, de hecho, cualquier cosa en la que un programa estaría interesado.

2.2 SOAP

El servicio web puede ser dividido aproximadamente en dos grupos, los basados en SOAP (SOAP-based), y los de estilos REST (REST-style). La distinción no es tan marcada ya que un servicio basado en SOAP que es entregado sobre HTTP, es un caso especial del servicio de estilo REST. SOAP originalmente significaba por sus siglas en inglés Simple Object Access Protocol , pero por fortuna, ahora su significado se acerca mas al de protocolo Service Oriented Architecture (SOA). Desmontar SOA, no es banal, pero un punto es indiscutible, lo que sea que SOA es, los servicios web

juegan un rol central en el enfoque de desarrollo y diseño de software SOA. SOAP, ya no seguirá siendo oficialmente un acrónimo, pero sigue estando separado de SOA. Por ahora SOAP es simplemente un dialecto de XML (EXtensible Markup Language) en el cual se documenta los mensajes (Kalin, 2012). La figura 2.3 ilustra la arquitectura.

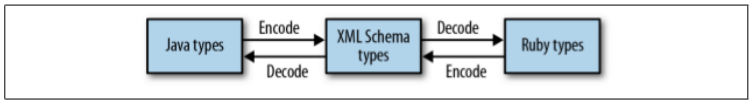

Figura 2.3: Arquitectura de un servicio web basado en SOAP. Tomado de (Kalin, 2012).

2.2.1 Esquema de SOAP

La dificultad en posibilitar una conversación, viene del hecho de que un nombre simple de datos como, por ejemplo, $START\_DATE$, no dice mucho a un programa. Si el Programa A, le dice Programa al B: "He aquí un $START\_DATE, 11/10/2005$", Programa B no va a ser capaz de utilizar la información. Eso es porque dicho mensaje no tiene contexto. El Programa B ni siquiera va a saber si $START\_DATE$ es una fecha. Se parece a una fecha, y tiene un nombre que indique que se trata de una fecha, pero no hay nada aquí que diga con certeza que se trata de una fecha. Incluso si es una, el programa B no puede estar seguro si se encuentra en formato Américano donde primero se coloca el mes, y luego el día, y el año (mm / dd / aaaa), ó en formato europeo, primero el día, luego el mes y año (dd / mm / aaaa). Incluso si el programa B hubiese pedido a Programa A "envíenme el $START\_DATE$" Programa B no estaría seguro acerca del mensaje que acaba de recibir. Lo que estos dos programas necesitan son algunas definiciones que pueden compartir. Por suerte, se cuenta con XML. El primer paso en la solución del problema es utilizar XML para especificar un lenguaje de definición de datos. Tal lenguaje se llama XSD (XML Schema Definition)

2.2.2 Componentes de SOAP

SOAP (Simple Object Access Protocol) es el protocolo de mensaje que los servicios web utilizan para hablar unos con otros, aunque no se limita a utilizar dichos servicios.

Fue inventado por Microsoft para hacer más fácil la construcción de software con Herramientas de desarrollo de Microsoft para interactuar con otros software. Es sencillo y flexible y puede ser utilizado por cualquiera de los dos programas que desean intercambiar mensajes.
Un mensaje SOAP tiene hasta cuatro componentes, a saber:

- **El Sobre:** rodea el contenido del mensaje y lo identifica como un mensaje de SOAP en lugar de cualquier otro tipo de mensaje.
- **La cabecera:** contiene las extensiones definidas por el usuario a SOAP. Estos pueden incluir actividades adicionales, como la autenticación por motivos de seguridad, que van más allá de los servicios que un programa pueda ser capaz de proporcionar para otro. Muy a menudo no hay una sección de encabezado de un mensaje SOAP.
- **El cuerpo:** contiene el mensaje, que es probable que sea una solicitud de un servicio o una respuesta, y es probable que incluyan datos.
- **El fallo:** llamada así sin rodeos , esto es una respuesta a un mensaje SOAP que genera un error, informando al remitente de cual error era. En la mayoría de los mensajes, no habrá sección de fallo. La figura 2.4 ilustra de mejor manera la estructura del sobre.

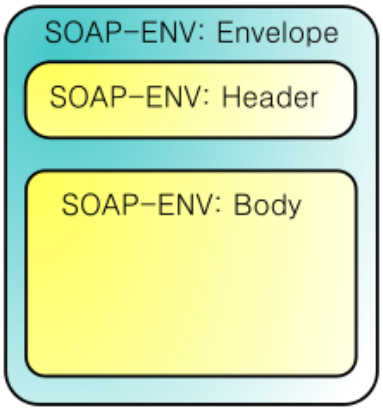

Figura 2.4: Ilustración de un sobre SOAP. Tomado de (Sokpop, 2009)

2.2.3 WSDL

WSDL es un dialecto de XML usado para describir el servicio web como una colección de "puntos extremos" de acceso capaz de intercambiar mensajes en un procedimiento, o también en modo documento. Un documento WSDL es una una suerte de "receta" utilizada para automatizar los detalles involucrados en la comunicación de aplicación-aplicación.

Todo lo que esté enmarcado dentro de un archivo WSDL es abstracto; se trata tan solo de la definición de parámetros y restricciones acerca de como la comunicación debe ocurrir en tiempo de corrida. La implementación de un servicio web debe adherirse a las pautas definidas por el archivo WSDL, aunque tiene cierta flexibilidad para algunos casos específicos. WSDL también provee la capacidad de definir ataduras que anexen un conjunto abstracto de definiciones de mensajes para un protocolo en concreto o para un formato de dato. WSDL establece extensiones predeterminadas para SOAP1.1, HTTP, GET, POST y MIME.

2.3 REST

Roy Fielding acuñó el acronimo REST (REpresentational State Transfer) en su tesis doctoral. En el capitulo 5 de dicha tesis se establecen los principios para lo que se ha ido conociendo como estilo REST (REST-Style) o servicio web RESTful (Kalin, 2012). REST y SOAP son muy diferentes. SOAP es un protocolo de mensajería, mientras que REST es un estilo de arquitectura de software para sistemas hipermedia distribuidos, es decir, sistemas en los que el texto, gráficos, audio y otros medios de comunicación se almacenan en una red e interconectan a través de hipervínculos. La World Wide Web es el ejemplo obvio de este sistema. Como el enfoque es el de servicios web, la World Wide Web es el sistema hipermedia distribuido de interés. En la Web, HTTP es a la vez un protocolo de transporte y un sistema de mensajería porque las solicitudes y respuestas HTTP son mensajes. Las cargas útiles de mensajes HTTP se puede escribir mediante el sistema de tipo MIME (Multipurpose Internet Mail Extensions). MIME posee tipos tales como text/html, application/octet-stream y audio/mpeg3. HTTP también provee códigos de estatus de respuesta para informar al solicitante acerca de

si su solicitud tuvo éxito, o de lo contrario porque no fue exitosa. Las figuras 2.5 y 2.6 muestran los estatus de HTTP con sus significados y la representación de un sistema RESTful respectivamente.

Status code	In English	Meaning
200	OK	Request OK
303	See Other	Redirect
400	Bad Request	Request malformed
401	Unauthorized	Authentication error
403	Forbidden	Request refused
404	Not Found	Resource not found
405	Method Not Allowed	Method not supported
415	Unsupported Media Type	Content type not recognized
500	Internal Server Error	Request processing failed

Figura 2.5: Tomado de (Kalin, 2012)

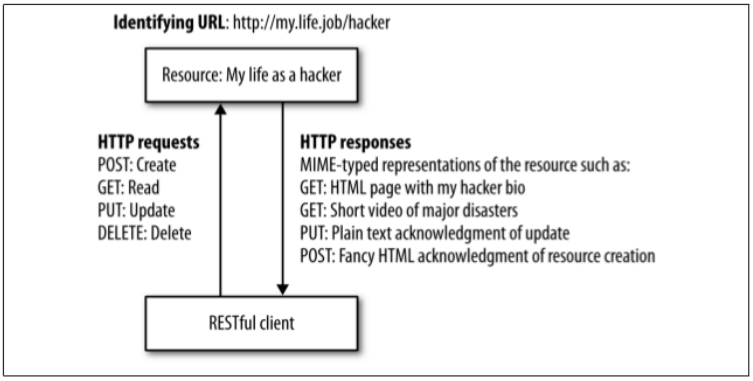

Figura 2.6: (Kalin, 2012)

2.4 Arquitectura orientada a servicios

Muchas definiciones de arquitectura orientada a servicios (SOA) se han ofrecido. Algunos lo ven como una arquitectura de estilo técnico que proporciona los medios para integrar sistemas dispares y descubrir las funciones de negocio reutilizables. Otros, sin embargo, tienen una visión mucho más amplia:

> "Una arquitectura orientada a servicios es un estilo de diseño que guía todos los aspectos de la creación y utilización de servicios de negocio a lo largo de su ciclo de vida (desde la concepción hasta su retiro)"
>
> "Arquitectura Orientada a Servicios (SOA) es un paradigma para organizar y utilizar capacidades distribuidas que pueden estar bajo el control de diferentes dominios de propiedad".

Estos puntos de vista sugieren que SOA es un paradigma de diseño o metodología en el que "las funciones de negocio" se enumeran como servicios, organizados en dominios lógicos, y de alguna manera gestionados durante su ciclo de vida. Mientras que SOA puede ayudar al personal de negocio articular sus necesidades de una manera que les resulta más natural que, por ejemplo, análisis orientado a objetos, todavía hay muchas maneras de implementar los servicios.

2.5 OpenAM

OpenAM es un manejador de acceso de sistemas de software. Dicho producto integra servicios de autenticación, autorización, SSO (Single Sign On)y protocolos basados en estándares abiertos de federación que provee SSO entre dominios de negocios distintos, derecho a uso de servicios y seguridad de servicios web. Sobre todo los usuarios serán capaz de construir una solución comprensible para proteger los recursos de las redes previniendo accesos no autorizados a servicios web, aplicaciones y contenido web y asegurando la identidad e integridad de los datos.
OpenAM ofrece una completa solución para asegurar tanto aplicaciones web como servicios web. Se puede reforzar una política de seguridad comprensible para aplicaciones y servicios web a través de compañías, en vez de tener que depender en desarrollos que surjan de forma rápida (ad hoc) para asegurar los servicios en la medida en que se van desarrollando. OpenAM es una aplicación Java que lo hace sencillo de desplegar en cualquier plataforma de sistema operativo o contenedor, ya que soporta un amplio rango de sistemas operativos y contenedores de servlets (Thangasamy, 2011).

2.5.1 Servicios OpenAM

Todos los servicios ofrecidos por OpenAM, son desplegados sobre el protocolo HTTP. Los usuarios acceden a ellos usando interfaces apropiadas. También expone una amplia librería de interfaces de programación de aplicaciones (API por sus siglas en ingles) e interfaz de proveedor de servicios (SPI por sus siglas en ingles), por las cuales los usuarios pueden lograr la funcionalidad deseada. Estos servicios desarrollados por OmenAM generalmente contienen tanto componentes de clientes como componentes del lado del servidor. El componente del servidor es un simple servlet de Java desarrollado para recibir peticiones XML y a su vez retornar respuestas XML. La aplicación web *openam.war*, enmarca todos los servicios y configuraciones de herramientas asociadas que se requieren para entregar la funcionalidad de OpenAM. El componente del lado del cliente lo provee el API de Java, y en algunos casos API de C. Esto permite aplicaciones remotas y otros servicios para comunicarse con y consumir una funcionalidad en particular.

Cada servicio usa su propio framework para obtener datos de usuario y servicios para el mismo y a su vez proveerlo a otros servicios OpenAM. El framework de OpenAM todos los frameworks de estos servicios para formar una capa que sea accesible a todos los componentes de los productos y plugins como los que se muestran en la figura 2.7 se puede observar como está estructurado.

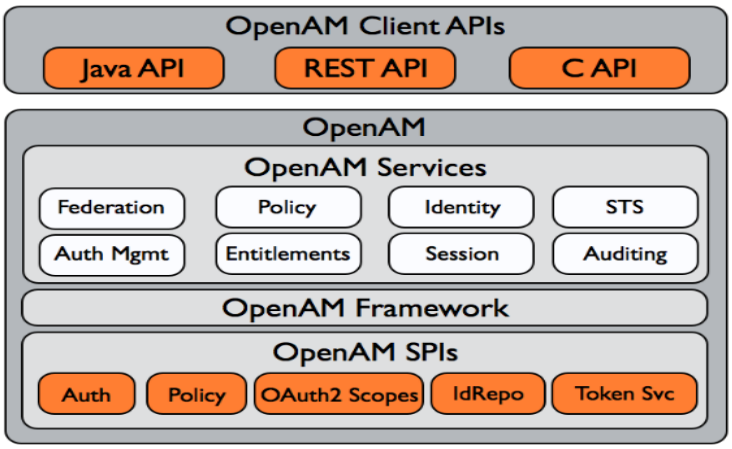

Figura 2.7: Arquitectura de OpenAM. (Craig, 2013)

2.5.2 Federación

Federación es un proceso que establece métodos basados en estándares para compartir y manejar datos de identidades y establecer Single Sign-On a través de dominios de seguridad y organizaciones. Permite a organizaciones ofrecer una variedad de servicios externos a socios u otros usuarios así como también sistemas empresariales hacia divisiones internas de departamentos. Formando "relaciones de confianza" vía dominios seguros permite a una organización integrar aplicaciones ofrecidas por diferentes departamentos o divisiones dentro del organismo, así como establecer relaciones con socios de negocios que ofrezcan servicios complementarios.
OpenAM ofrece un framework abierto y extensible para federación de identidades y servicios web asociados, que resuelven los problemas de habilitación de identidad, servicios web discovery e invocación, seguridad y privacidad. La federación han sido desarrollada sobre los siguientes estandar:

- Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) 1.1 and 1.2.
- OASIS Security Assertion Markup Language (SAML) 1.0 and 1.1 .
- OASIS Security Assertion Markup Language (SAML) 2.0 .
- WS-Federation (Passive Requestor Profile).

SAML 2.0 se ha convertido en el estándar para la federación SSO, y esto se ve reflejado en el creciente números de compañías que lo soportan. Por ejemplo Google Apps y Salesforce hacen uso de SAML 2.0 como su protocolo de elección para SSO.

2.5.2.1 Fedlets

Un fedelet OpenAM, es una pequeña aplicación web que puede hacer federación a la aplicación proveedor de servicio, con OpenAM actuando como el proveedor de identidades. Los fedlets no requieren una instalación completa de OpenAM junto con la aplicación web que haya sido desarrollada, sino que puede redirigir hacia el single sign on, y obtener las aserciones de SAML.

2.5.3 SAML 2.0

SAML 2.0 (Security Assertion Markup Language 2.0) es una version del estándar SAML para intercambio de autenticación y autorización de datos entre dominios seguros. Como su nombre lo sugiere provee un lenguaje de marcado para representar aserciones de seguridad (que en este contexto podría verse como sentencias del lenguaje). SAML 2.0 es un protocolo basado en XML que usa tokens de seguridad conteniendo aserciones para pasar información del "principal" entre una autoridad, que en este caso es el proveedor de identidad y un consumidor SAML, que es el proveedor de servicio. SAML 2.0 habilita escenarios de autenticación y autorización web incluyendo SSO (single sign-on), lo cual ayuda a reducir las complicaciones administrativa de distribuir múltiples tokens para autenticación al usuario. Ver figura 2.8

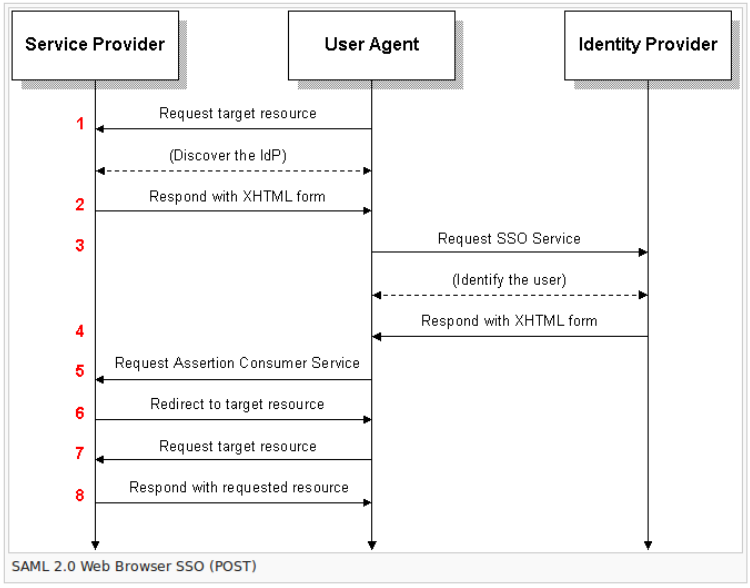

Figura 2.8: Diagrama de secuencia SAML. Tomado de (Trscavo, 2007)

2.5.4 Perfil SSO del navegador web

SAML 2.0 especifica un perfil SSO del navegador web que involucra al proveedor de identidad , el proveedor de servicio y al principal que ejerce a un usuario agente HTTP. El SP tiene cuatro ataduras de las cuales escoger y el IdP tiene tres, lo cual lleva a doce

escenarios de despliegues posibles, y del cual se describió uno en el apartado anterior a través del esquema mostrado.

2.6 Simulación Distribuida

La Simulación Distribuida, es la que permite conducir modelos de simulación a través de múltiples hosts o computadoras separados por redes, se ocupa de cuestiones que surgen de la distribución de un programa de simulación de eventos discretos en varias computadoras (Banks, 2004). La simulación paralela de eventos discretos se refiere a la ejecución en plataformas multi-procesador que contienen múltiples unidades de procesamiento central que interactúan con frecuencia, por ejemplo, miles de veces por segundo. La simulación distribuida se refiere a la ejecución de simulaciones en sistemas débilmente acoplados en los que las interacciones toman mucho más tiempo, por ejemplo, milisegundos o más, y se producen con menos frecuencia. Incluye la ejecución en computadoras distribuidas geográficamente a través de una red de área amplia, como la Internet.

2.7 Simulación de Eventos Discretos

La simulación de eventos discretos es la "imitación" de un proceso de operación o de un sistema del mundo real construido sobre la base del tiempo (Banks, 2004). Esta codifica el comportamiento de sistemas complejos como una secuencia de eventos ordenados y bien definidos. Una computadora paralela se utiliza con mayor eficacia y con menos dificultad de producir gran número de muestras estadísticamente independientes de un proceso estocástico. Esta aplicación de la computación en paralelo es de vital importancia para sacar conclusiones significativas a partir de modelos que incluyen números aleatorios, y por lo tanto es muy común en la práctica. En simulación de eventos discretos (DEVS), el funcionamiento de un sistema se representa como una secuencia cronológica de los acontecimientos. Cada evento tiene lugar en un instante de tiempo y marca un cambio de estado en el sistema.

2.7.1 Descripción de GALATEA

Como se ha mencionado en puntos anteriores, se busca que el desarrollo del nuevo sistema tenga como base la plataforma implementada en la Universidad de Los Andes, denominada GALATEA, cuyos creadores/colaboradores han sido: Uzcátegui Mayerlin, Dávila Jacinto y Tucci, Kay. En el siguiente fragmento se explica breve, pero concisamente en qué consiste la misma. Tomada de (Uzcátegui, 2011):

> "Galatea es una plataforma libre de código abierto para simulación de sistemas multi-agente que incorpora estrategias de simulación bien conocidas con la que cualquier modelista o simulista puede ensayar esas estrategias en problemas de simulación de sistemas complejos. La historia de GALATEA comienza mucho antes que se planteara formalmente el proyecto con ese nombre. En 1993, nuestro muy joven Centro de Simulación y Modelos, CeSiMo, propone un proyecto para explorar la reimplantación de la plataforma de simulación Glider sobre una plataforma orientada a objetos dando origen a un prototipo experimental. El problema del cambio estructural, inspirado por investigaciones en economía se había convertido entonces en uno de los objetivos de investigación fundamentales del CeSiMo y vendría a dictar también la pauta para Galatea. La noción de agente hizo su aparición en algunos reportes internos en los que se enfatizaba su importancia para modelar sistemas complejos como una economía nacional. En 1998 se planteó la posibilidad de integrar Glider con herramientas de inteligencia artificial para modelar agentes. En el 2000, un proyecto vendría a combinar aquel prototipo de 1993, con una teoría de agentes basada en lógica computacional que se planeaba integrar en una nueva teoría de simulación de sistemas multi-agentes. Allí nació Galatea. El logro fundamental para el proyecto, sin embargo, llegaría con las aplicaciones. En 2004, Galatea fue incorporada al banco de pruebas de un proyecto en biocomplejidad. Las lecciones aprendidas desde entonces, los aportes particulares del proyecto, así como los desaciertos y caminos futuros, son discutidos en este documento."

2.7.2 Diseño de GALATEA

La arquitectura de GALATEA está basada en objetos. Tanto los agentes como el simulador principal están desarrollado de acuerdo a dicho diseño (orientado a objetos, OOD), para apoyar la distribución, la modularidad, la escalabilidad y la interactividad como lo exige el especificación HLA (High Level Architecture). Está destinado a una plataforma flexible desde el punto de vista de la ingeniería de software (que es, podría decirse, inaccesible a los usuarios finales: los modelistas), pero en base a una familia de lenguajes amigables para el modelado con expresividad suficiente para permitir a los creadores de modelos describir un sistema multi-agente multi-sistema de una manera que hace factible su simulación. Se considera que esta posibilidad depende críticamente de las ventajas y desventajas del dominio y la aplicación específica. Por lo tanto, se permite que los modeladores describan sistemas en los que hay agentes, pero en el que no todo es un agente, y en el que las técnicas tradicionales de modelado de eventos discretos o continuos son lo suficientemente bueno para la mayoría de propósitos (tales como tratar con subsistemas que requieren muchos modelos agregados para hacer su simulación lo suficientemente factible).
GALATEA está escrito en Java, su árbol de directorios está básicamente conformada por siete carpetas:

- /bin
- /config
- /demos
- /doc
- /galatea
- /images
- /lib

A su vez que contiene archivos de invocación/configuración, los cuales corresponden a sistemas operativos distintos, “galatea.bat” y “galatea.sh”, para Windows y Linux

respectivamente.

En este momento GALATEA tiene ocho librerías / paquetes. Kernel incluye el motor de simulación de eventos discretos (planeador), el framekwork HLA (HLA), toolkits de eventos Continuo (gode) y simulación Multi-Agentes (gloria).Además está llevando a cabo desarrollos que incluyen código traducción Galatea-Java (gcompiler) y la interfaz gráfica de usuario (ggui) como parte del Entorno Integrado de Desarrollo y los kits de herramientas para los sistemas de información geográficas (ggis) y diseños arquitectónicos urbanos (gspaces), en la figura 2.9 se ilustra mejor a través del diagrama de clase el componente del motor de GLIDER.

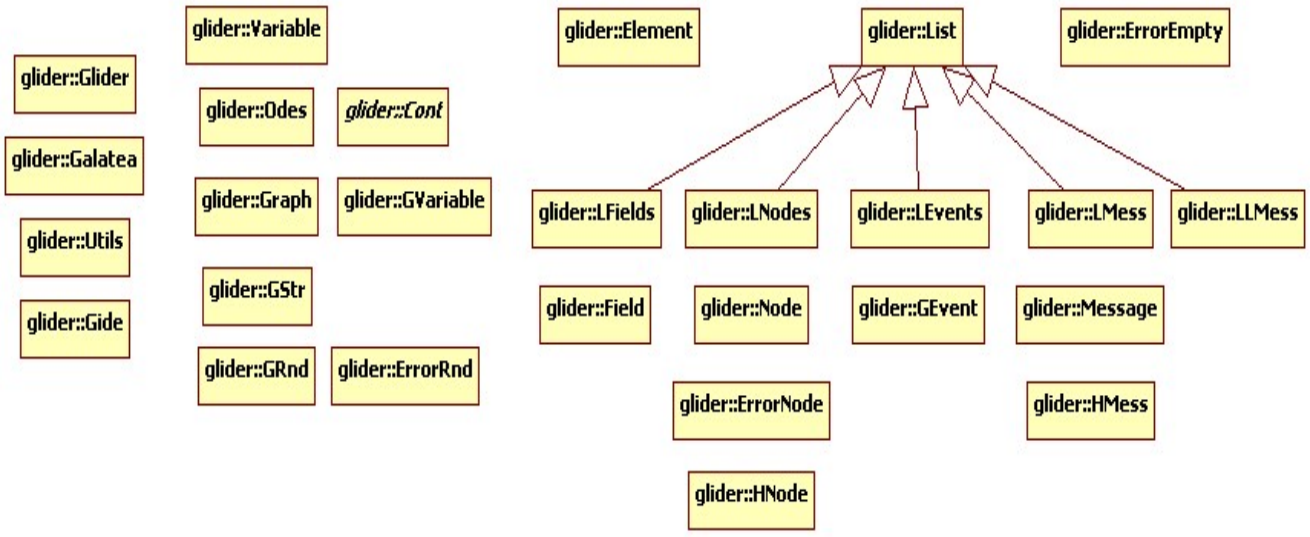

Figura 2.9: Glider. Motor simulador y clases básicas GALATEA

Esta biblioteca es parte de Mayerlin Uzcátegui S. Tesis Maestría en Ciencias de la Computación supervisado por Jacinto Dávila Q

Capítulo 3

Modelado y análisis de requerimientos

En este capítulo se especifican los requerimientos del sistema a desarrollar. Siguiendo la metodología desarrollo en espiral para los requisitos del sistema, y haciendo uso de UML que tienden a usar otros métodos de desarrollo, se instanciará lo que se explicó en el capítulo anterior acerca de las fases de lo que es el método de espiral, las cuales consisten en describir lo que se hizo en cada iteración/fase.

3.1 Iteración 1

En esta fase se planteó qué se requería, y cual sería el enfoque para el desarrollo. El Dr Jacinto, plasmó de que se trataba GALATEA, y la visión de como se habría de integrar con las otras herramientas que a lo largo de la fase de desarrollo se fuese estableciendo apropiada. Otros de los puntos en esta iteración consistió en la elección de la metodología a usar, y tal como se explicó en el capitulo 1, después de evaluar cuidadosamente se toma la decisión de que el desarrollo en espiral es el que mejor se adapta a la característica de este proyecto. Siguiendo esa linea se expone las tareas correspondiente:

3.1.1 Determinar Objetivos.

- El objetivo en esta tarea se enmarcó la búsqueda en la literatura de hallar con qué se iba a trabajar. Por ejemplo, el poco conocimiento por parte de este servidor acerca de , en qué consistía un servicio web, tomó un tiempo corto pero aun así importante para aclarar lo que esta herramienta en verdad suponía.

- Otra tarea para obtener un objetivo fue la de visualizar como sería la interacción y el funcionamiento del sistema una vez empezara a funcionar. Por ejemplo, poder realizar corridas de modelos de simulación desde un portal web, vía servicio web, sin tener que realizar mayores instalaciones a la computadora que solicita dicho modelo.

- Iniciar la búsqueda de una tecnológica que se encargara de la seguridad del sistema. Aun cuando existen incontables mecanismos de donde obtener lo que se necesitaba, se tuvo que ser minucioso y prudente, ya que como se verá mas adelante, la elección incorrecta, conllevaría a invertir mucho mas tiempo de lo anticipado era necesario en este objetivo. Esta experiencia mas allá de ser consumidora, también fue enriquecedora ya que a través los intentos fallidos, se logró aprender cosas nuevas que a futuro sí podrían resultar útiles en otros proyectos.

3.1.2 Análisis del riesgo.

- Uno de los principales riesgos a sopesar fue la nula interacción y desconocimiento con un servicio web por quien escribe este documento.

- La escogencia de una herramienta adecuada tanto a nivel de la tecnología usada para desarrollar el servicio web, como para el manejo de la seguridad, todo eso aunado a que alguna mala elección podía comprometer parte del proyecto.

- El poco conocimiento que un principio se tenía (el autor de este texto) acerca del funcionamiento de GALATEA, y por lo cual conllevaba un tiempo a invertir en como se habría de integrar dicho sistema a un servicio web.

3.1.3 Desarrollar y probar.

- Una de las primeras pruebas que se hizo fue las herramientas con la que ya se contaba conocimiento, como en este caso, la instalación de java, eclipse, apache tomcat, y como novedad también se probó Axis2 como medio para publicar servicios web.

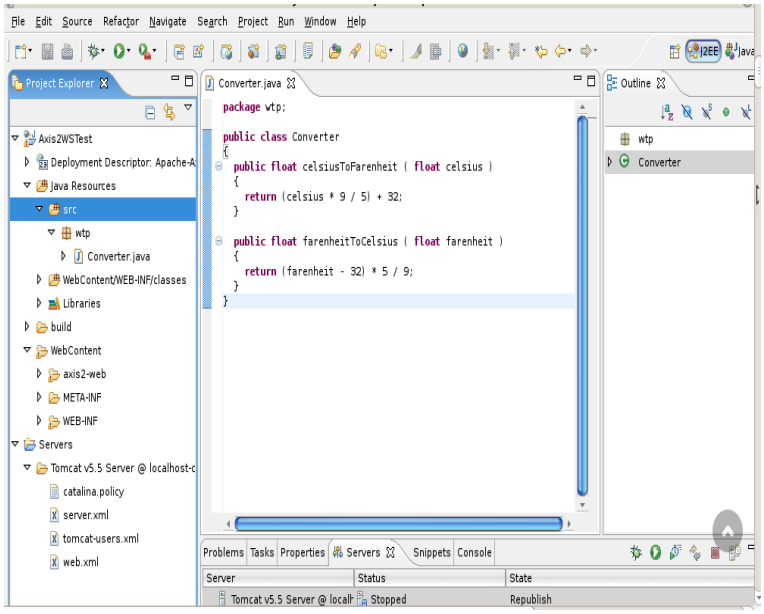

Figura 3.1: Demo de un servicio web.

3.1.4 Planificación

Al finalizar esta iteración se concluyó que era importante empezar a enfocarse en la herramienta para la seguridad, que en este caso OpenAM. A su vez también se procuró con empezar el proceso de establecer cual sería la interfaz de usuario y la tecnología que la proporcionaría. Así también se obtuvo corridas de muestras para lo que constituiría un servicio web. Dicha muestra está caracterizada por un pequno servicio web que convierte temperatura Farenheit a Celsius, y viceversa. Ver

3.2 Iteración 2

3.2.1 Determinar los objetivos

- Uno de los productos definidos a obtener, era el de dominar el funcionamiento de Openam, cuya herramienta requerida para el proyecto consistía en una fedelet. La fedelet es una pequeña aplicación web que funge de federación para el proveedor de servicio (esto es, los servicios que proporciona GALATEA) y que actúa con OpenAM como proveedor de identidad.

- Con el propósito de evaluar varias opciones en cada una de las áreas y componentes que tendría el sistema, se añadió otra novedad en esta fase, que fue la inclusión de Virtuoso, el cual consistía en un servidor múltiple que se establecería como una capa adicional dentro de la arquitectura original (una suerte de middleware) a través de la cual se tendría servicios como sistema gestor de base de dato relacional (RDBMS), sistema de gestor de base de dato orientada a objeto (ORDBMS), base de datos virtual, marco de descripción de recursos (RDF: Resource Description Framework), XML, y aplicación Web.

3.2.2 Análisis del riesgo.

- Al estudiar las posibles amenazas, y en aras de reducir las probabilidades de falla en el proyecto, como se mencionaba en el apartado anterior, se halló alternativa a OpenAM. Tras una ardua tarea de pruebas e intentos fallidos con el manejo Virtuoso, se optó por no hacer uso del mismo, pues a pesar de lo enriquecido que era dicho software, y de ofrecer las funcionalidades que abordaban los requerimientos necesarios para el proyecto, demostró tener (a los ojos de este humilde servidor, y otro tesista que post grado) cierta complejidad en su instalación y uso, lo cual propició que se desestimara.

3.2.3 Desarrollar y probar.

- En esta fase se experimentó contratiempos con el uso de fedlet que ofrecía OpenAM, por alguna razón no ejecutaba de manera correcta.Sin embargo luego

de grandes esfuerzos, se logró obtener un demo que pudiera comprobarse. De cualquier manera no se desestimó por completo la elección inicial de OpenAM, sin embargo sí se hizo un reenfoque de esfuerzo hacia otras áreas del proyecto. En la siguientes imagen una muestra de una version "test" de fedlet.

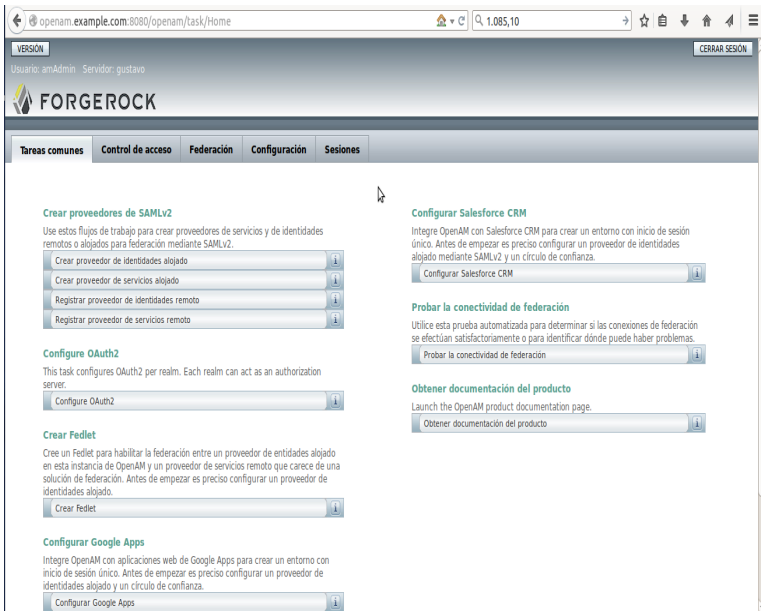

Figura 3.2: Panel de Administrador OpenAM.

- Finalmente, como parte de este fase en la iteración, se comenzó a esbozar e implementar con Jsp como parte de la interfaz para probar a un nivel mayor las funcionalidades del servicio integrado con el simulador.

3.2.4 Planificación

Para la siguiente iteración se estipuló que se debía tener un prototipo de modelo de simulación GALATEA corriendo en una interfaz web, a fin de probar su funcionamiento correcto, incluyendo la integración de todos los componentes con los que se estuvo trabajando para obtener un prototipo preliminar

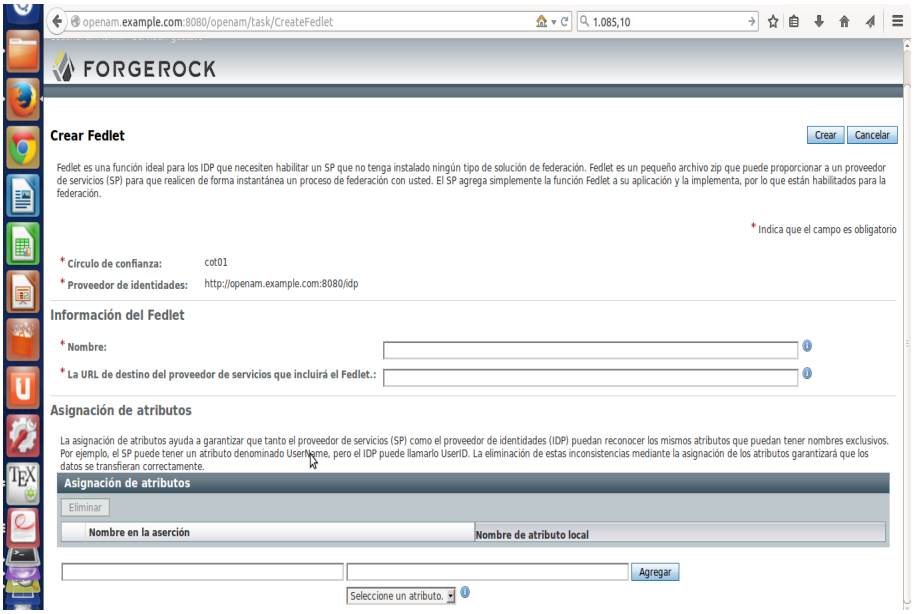

Figura 3.3: Creando fedlet OpenAM.

Figura 3.4: Interfaz demo de fedlet OpenAM.

3.3 Iteración 3

En esta ultima iteración se lograron varios hitos en varios frentes del proyecto, como el de lograr obtener una prueba exitosa de fedlet, así como la primera corrida del modelo

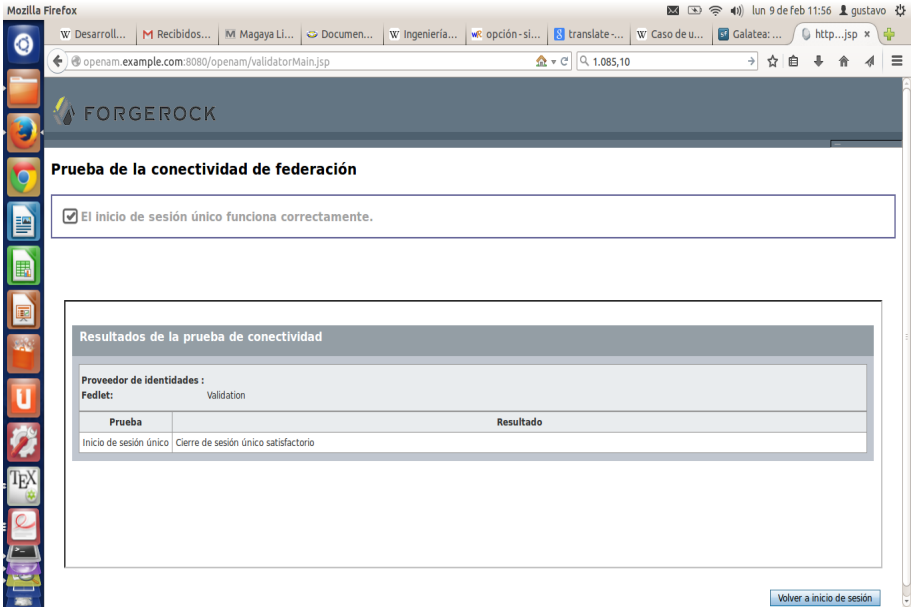

Figura 3.5: Prueba exitosa de fedlet.

GALATEA desde un entorno amigable.

3.3.1 Determinar los objetivos

- Cabe mencionar que en este set de actividades se empezó a sopesar el uso de maven como manejador de proyecto, esto con la finalidad de simplificar la creación de modelos y mantener un árbol de directorios mas estructurado y ordenado, al igual que podría hacerlo mas escalable para futuras adiciones. Sin embargo esta elección no se logró concretar a falta de tiempo y mayor experiencia, pero deja la puerta abierta para futuros trabajos e integración.

- Se obtuvo una finalmente una ejecución de fedlet en OpenAM, lo que abrió el camino para finalmente integrarlo al proyecto.

3.3.2 Análisis de riesgo

- Uno de los riesgos identificados, ha sido el experimentar con una nueva tecnología como maven, la cual supone bastantes beneficios. Sin embargo, cuando se evaluó

el costo en términos de tiempo, se decidió dejar a un lado esta opción debido a que se corría el riesgo de no completar los plazos de tiempo fijado para le entrega.

3.3.3 Desarrollar y probar.

En esta fase se obtuvieron la interfaz de usuario, así como muestras de pruebas de la corrida de GALATEA. Cabe destacar que la interfaz consiste en un gestión de acceso al usuario el cual introduce su usuario y contraseña. Dicha gestión está precedida por la funcionalidad de registrarse vía formulario, y es almacenado en el gestor de base de datos MySQL, a su vez diseñada de manera básica. El fin ultimo es lograr integrar mediante el framework ofrecido por OpenAM la capa de seguridad integrada, necesaria para obtener la robustez que se requiere para la corrida de los modelos. En la figura se vislumbra las trazas y estadísticas obtenidas por la corrida del modelo GALATEA.

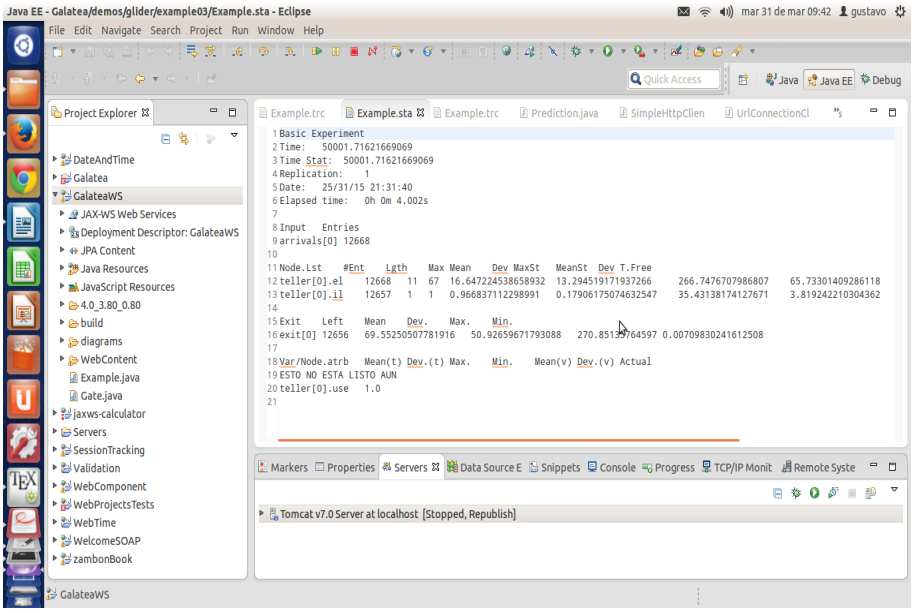

Figura 3.6: Estadísticas de Galatea.

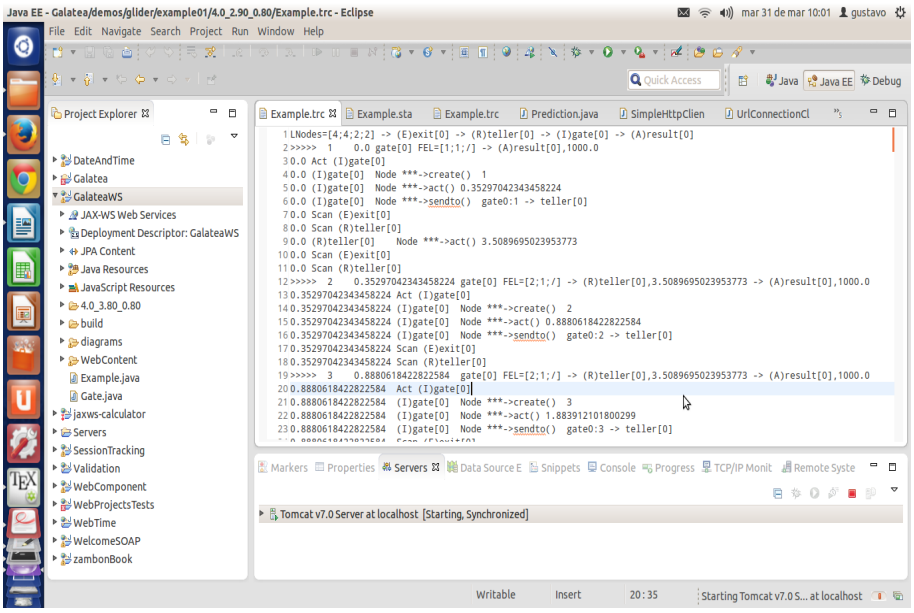

Figura 3.7: Trazas de corrida de Galatea.

Figura 3.8: Interfaz De Usuario Galatea.

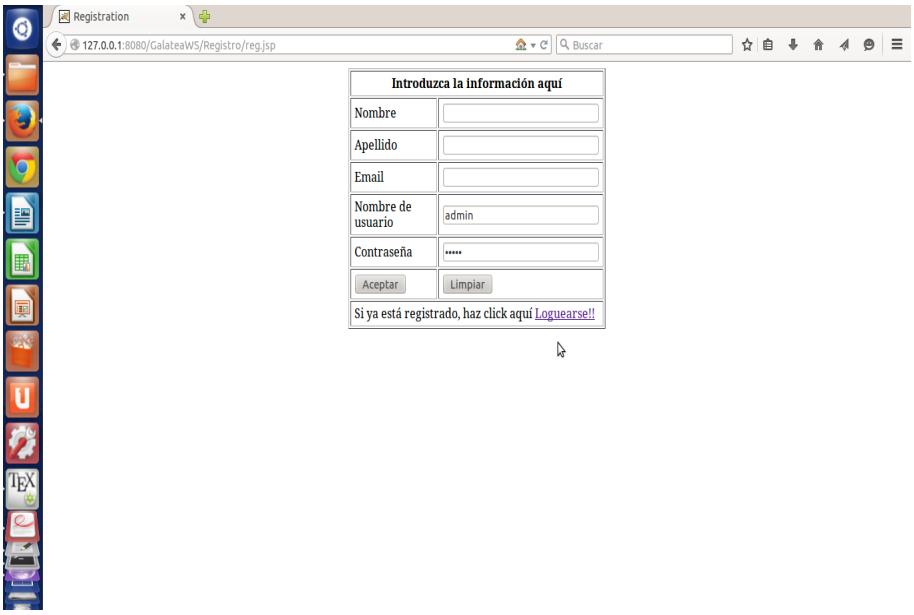

Figura 3.9: Registro De Usuario Galatea.

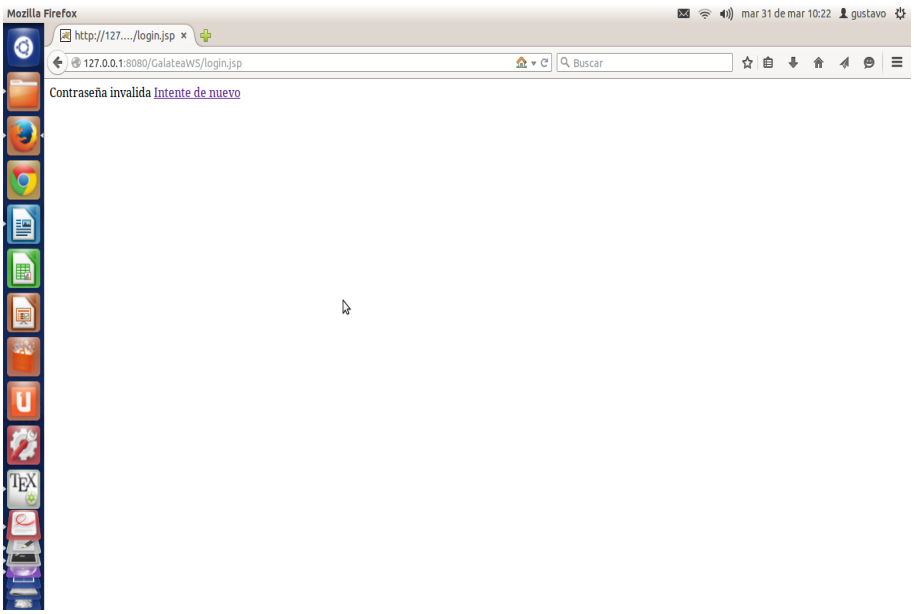

Figura 3.10: Registro invalido Usuario Galatea.

Capítulo 4

Diseño, implementación y prueba del Sistema

En éste capítulo se explicará de manera detallada el proceso mediante el cual se llevó a cabo el diseño y la implementación del sistema. En virtud de que el desarrollo consistió de una serie de adaptaciones de software preexistente que hubo que instalar y verificar, las pruebas del sistema se hicieron concurrentemente y por tanto, se presentan de la misma manera. Ciñendose al modelo de ciclo de vida del software que acompaña al método de desarrollo en espiral,en cada fase de iteración se distribuyó en varias tareas, que al final de la iteración generaría un "mini" producto, que bien pudiera ser visto desde la perspectiva informática como un componente de software que formaría parte del sistema en conjunto.

4.1 Selección del sistema a utilizar

4.1.1 Gestión del servicio y usuario

REStful API forma parte de las características que OpenAM ofrece y son necesarias para los requerimientos del proyecto. Este ofrece las siguientes operaciones para manejo de acceso de identidades.

- Autenticación (login)

- Logout
- Información de cookie
- Recuperación de atributo de token
- Validación de token
- Autorización
- Autorización OAuth 2.0
- OpenID connect 1.0
- Reseteo de contraseñas olvidadas
- Manejo de identidades (creación, lectura, actualización y borrado de identidades)
- Manejo de esfera de seguridad (realm)
- Despliegue de panel de administrativo

Todas estas vienen de fuera de la caja (out of the box), es decir funcionalidades prácticamente listas para su uso, con virtualmente poca o ningún ajuste o modificación. Nuevamente debido al alcance de este proyecto, no serán necesarios hacer usos de todas ellas, mas sin embargo, queda la puerta abierta para que en trabajos futuros, si se decidiese explorar u ahondar en ellas, poder extraer todo su potencial en función de la escala de nuevas características que el nuevo sistema pudiera empezar a tener.

4.1.2 Especificación del sistema

En función de ilustrar mejor el uso de las herramientas que conforman el sistema, de despliegan un par de diagramas que ilustran de manera muy básica pero muy practica a su vez la esencia de lo que se buscó a lo largo del desarrollo de este proyecto. No está demás hacer mención que el grado de complejidad inherente al aprendizaje y manejo de OpenAM supuso grandes retos en cuanto a la integración de todo el sistema en su conjunto.

Tomando que GALATEA cumple con ser el servicio dentro de los roles que se ha estado desarrollando se muestra los pasos (algoritmo si se quiere) de un despliegue de fedlet de OpenAM.

1. Antes de crear un fedlet es necesario crear el proveedor de identidades local en caso de haber creado uno previamente.
2. En la consola de tareas administrativas de se cliquea sobre Crear Fedlet.
3. OpenAM posee un parámetro denominado “circulo de confianza” (Circle of Trust)
4. Se deberá notar que el circulo de confianza incluye el proveedor de identidad almacenado (a diferencia de si se ha creado uno remoto)y que el parámetro “Proveedor de Identidad” estará establecido precisamente en proveedor de identidad almacenado.
5. Se identifica con un nombre a la fedlet y se establece la URL de destino. Se puede utilizar el URL:

```
http://openam.example.com:8080/fedlet
```

   tanto como nombre así como de destino de la URL
6. Se hace click sobre el boton “Crear”, para generar el archivo Fedlet.zip.
7. Finalmente, se descomprime el archivo Fedlet.zip y se traslada al archivo donde se despliegan las aplicaciones web a través de apache tomcat:

```
/usr/local/apache-tomcat-7.0.53/webapps/
```

Nótese que para efectos de el conjunto, OpenAM puede ser vista si se prefiere como una caja negra que se encarga del manejo de proveedor de servicio e identidades a través de las fedlets. A continuación un modelo mas elaborado en diagrama de caso de

uso acerca de lo que es, y también de las funcionalidades agregadas que podría tener a futuro trabajando con los diseñadores iniciales de GALATEA. En cuanto a la parte de funcionalidades agregadas se indica que se refiere a la posibilidad de detener las corridas de un modelo en pleno proceso, y ajustar los parámetros de la misma y reanudar de manera que se pudiera vislumbrar como pudiera afectar el cambio en dichos parámetros en cuanto al resultado de la muestra. Esto fue una teoría que se planteó con el doctor Jacinto para futuros trabajos pero que se decide incorporar en el presente trabajo por motivo de era buena idea plasmarlo.

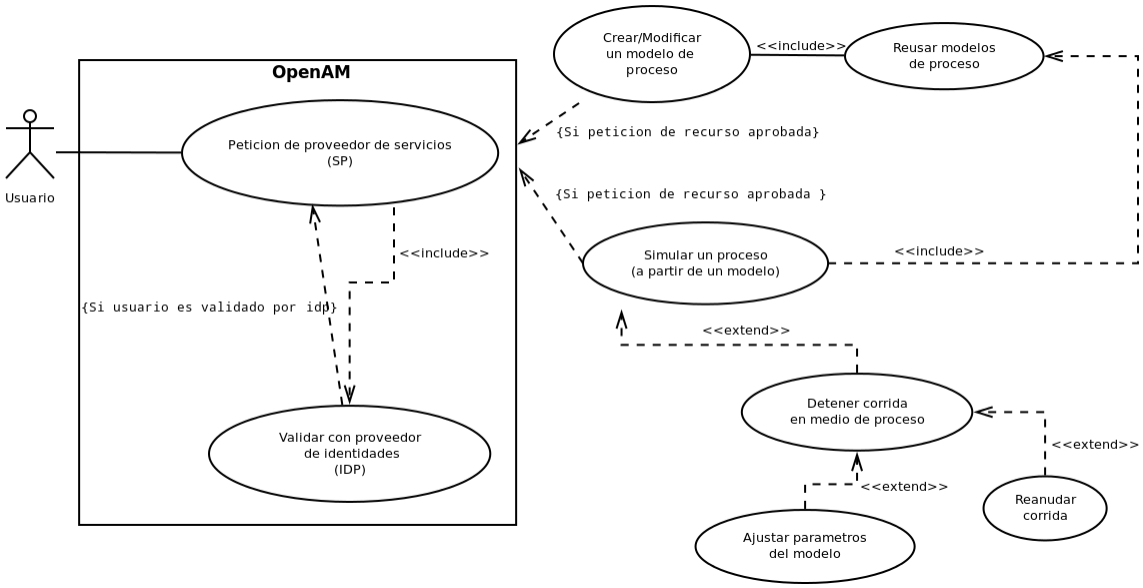

Figura 4.1: Diagrama mas detallado de interacción

4.2 Arquitectura de la interfaz de usuario

En esta sección se esboza de manera muy sencilla lo que constituye la arquitectura de la interfaz de usuario, haciendo referencia a capitulo anteriores y recordando que se hizo uso de jsp como tecnología de elección para la GUI

Los siguientes pasos explica como el servidor web crea la pagina que sirve como plataforma y que invoca al servicio web de GALATEA:

1. Como con una pagina normal, el navegador envía una peticion HTTP al servidor web, pero la extension terminará en jsp.

2. El servidor web, no es un servidor normal, sino un servidor java, con las extensiones necesarias para identificar y manejar servlets. El servidor web reconoce la que la petición HTTP es para una pagina jsp y la pasa al motor jsp.

3. El motor jsp carga la pagina jsp desde el disco y la convierte en un java servlet. A partir de este punto, el servlet es indistinguible con respecto a otro servlet desarrollado directamente en java en vez de en jsp, aunque el código java generado automáticamente para un servlet jsp no siempre es fácil de leer, no se deberá modificar manualmente.

4. El motor jsp compila el servlet en una clase ejecutable y pasa la petición original hacia otra parte del servidor web denominado motor de servlet. Hay que hacer notar que el motor jsp solo convierte solo convierte la pagina jsp a una pagina java, y recompila el servlet si halla que la pagina jsp ha sido modificada. Esto hace el proceso mas eficiente que otros lenguajes de tipo scripting como PHP, por tanto haciéndolo mas rápido.

5. El motor del servlet carga la clase, y la ejecuta. Durante la ejecucion, el servlet produce una salida en formato html, el cual el motor servlet pasa al servidor web dentro de la respuesta HTTP.

6. El servidor web pasa la respuesta HTTP al navegador.

7. El navegador web maneja dinámicamente la pagina html generada dentro de la respuesta HTTP tal como si fuera una pagina estática, de hecho, las paginas estíticas y dinámicas están en el mismo formato.

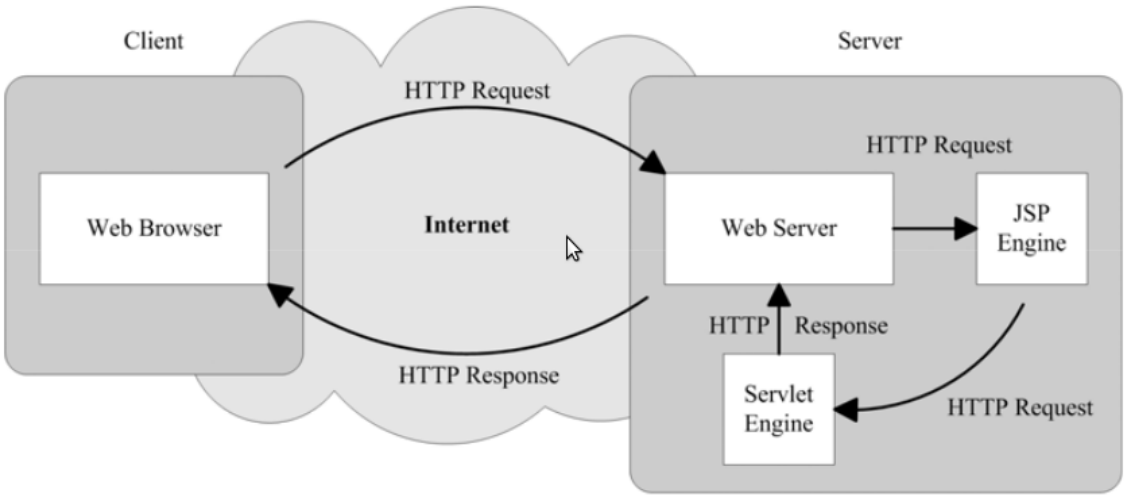

Figura 4.2: Arquitectura interfaz de usuario

Capítulo 5

Conclusión

La mayoría de las conclusiones obtenidas durante la elaboración del proyecto tienen que ver con aprendizajes acerca de la forma ideal de desarrollar software libre, a su vez que integrarlos de manera armónica, principalmente provenientes del software utilizado durante el proyecto y las trabas encontradas con el mismo. Del mismo modo, varias de estas conclusiones podrían ser vistas como experiencia para a los desarrolladores que trabajen con las tecnologías usadas, con la finalidad de ayudarles a mejorar sus productos de software, pero al mismo tiempo las lecciones aprendidas durante el desarrollo del proyecto como errores que se deben evitar en un futuro.

Por parte del proyecto, aún cuando hubo un trabajo enfocado hacia solucionar el problema descrito, el objetivo principal no se cumplió completamente, por lo que se acotó al no haberse realizado una implementación propia de un sistema sino una instalación de un par de productos de software que, se esperaba, pudiesen ayudar resolver parte del problema anteriormente planteado. En consecuencia, ambos productos de software no se integraron completamente debido al tiempo prolongado en el aprendizaje de varias de la herramientas que intervenían en el producto, principalmente OpenAM, que si bien es muy poderosa, su extensiva cantidad de funcionalidades, y su API de clientes propiciaron una prolongación mas allá de lo estimado.

- Se planteó utilizar OpenAM para permitir a los usuarios iniciar sesión al sistema utilizando las mismas credenciales, pero no se creó una plataforma centralizada

que una ambas plataformas. Además de esto, no se llegó a integrar esto con el servicio web, de manera que no se pudo comprobar a cabalidad la total operatividad de todo los componentes del sistema utilizados en conjunto.

Por otra parte, hubo varias enseñanzas durante el desarrollo del mismo, principalmente provenientes de los errores hallados con las aplicaciones utilizadas que vale resaltar. Los principales son:

- Al desarrollarse un software o sistema web que a su vez requiera de varios otros componentes, resulta importante distribuir un paquete que contenga todos o la mayor parte del software requerido, y que en lo posible tenga la propiedad de configurarse de manera fácil (considero que un repositorio SVN aunado con Maven podría ser muy apropiado). Esto resulta importante debido a que no siempre el usuario del software tendrá los conocimientos necesarios para poner algunos productos de software a ejecutarse o para depurar cualquier error que aparezca en el proceso. Es por ello que como parte de este documento, en el apartado de apéndices se plasman las instrucciones y pasos a seguir que usó el autor para lograr obtener el funcionamiento de las herramientas requeridas. Ciertamente, y al igual que como en la vida, se prevee que todo este conocimiento, experiencia y aprendizaje no ha sido en vano, pues deja sentada las bases para trabajos futuros de esta índole, y facilitará el camino a otros, de manera que ya no empiecen desde cero, sino que mucho de lo aquí ganado será un gran aliciente para finalizar lo que haya quedado pendiente, así como enriquecer y ampliar lo existente.

- Resulta esencial la elección del software “correcto”, y por esto se refiere a software que primero que todo, cubra los requerimientos que un proyecto establece. Adicionalmente, que se encuentre soportado actualmente por desarrolladores que ofrezcan documentación e información en caso de algún bug o cualquier otro inconveniente.Todo esto es muy importante pues no sería viable el desarrollo de algun software si parte de las herramientas se encuentran descontinuadas.

5.1 Apéndices

A continuación se presenta una serie de guías que han resultarán importantes para replicar en otros equipos el trabajo que en este proyecto se plasma.

5.1.1 Instalación de Java

De seguro existen otras formas de realizar el proceso de instalación, pero esta es la que mejor le ha funcionado al autor a lo largo de los años.

1. Descargar java de: http://java.sun.com/
   Ver "Downloads->Java SE->Java", el archivo a descargar tiene un nombre similar a:

```
jdk1.7.0_45.tar.gz
```

   – Los siguientes pasos deben hacerse como super-usuario (root) –

2. Copiar o mover, usando el comando cp o mv, el archivo descargado al directorio /usr/local/src

3. Ubicarse, utilizando el comando cd (Ej: cd /usr/local/src), en el directorio /usr/local/src

4. Descomprimir el archivo "jdk1.7.0_45.tar.gz" mediante el comando:

```
tar zxvf jdk1.7.0_45.tar.gz
```

5. Mover el directorio resultante "jdk1.7.0_45" al directorio /usr/local/:

```
mv jdk1.7.0_45 /usr/local/
```

6. Asignar permisos de ejecución al directorio que se ha descomprimido:

```
chmod 755 -R jdk1.7.0_45
```

7. Ubicarse en el directorio /usr/local (Ej: cd /usr/local)
8. Hacer un enlace simbólico java al directorio jdk1.7.0_45, por ejemplo:

```
ln -s jdk1.7.0_45 java
```

9. Para que se pueda utilizar el comando java en la consola desde cualquier ubicación es necesario poner la ruta del ejecutable de Java en la variable de entorno PATH:

```
export PATH=/usr/local/java/bin:$PATH
```

10. Luego de poner la ruta en el PATH debe ser posible ejecutar el comando java y obtener como resultado:

```
gustavo@gustavo: $ java -version
java version "1.7.0_45"
Java(TM) SE Runtime Environment (build 1.7.0_45-b18)
Java HotSpot(TM) 64-Bit Server VM (build 24.45-b08, mixed mode)
```

y además:

gustavo@gustavo: $ which java /usr/local/java/bin/java

– Los siguientes pasos deben hacerse como un usuario normal (NO como root) –

11. Los cambios realizados en el PATH en el Paso 8 serán válidos sólo para el terminal en el que se ejecuta la instrucción export en cuestión. Si se desea que el PATH sea correcto en cualquier terminal sin necesidad de hacer el export para cada consola abierta, entonces es necesario configurar los archivos de inicio del shell, estos, si se está utilizando bash o sh suelen ser los siguientes:

.bashrc

.bash_profile

En alguno de esos archivos añadir:

export JAVA_HOME=/usr/local/java

PATH=$JAVA_HOME/bin:$PATH

12. Luego del paso 11, debe ser posible abrir cualquier terminal y obtener los mismos resultados que en el **paso 9**.

5.1.2 Instalación de eclipse

Como se indicó al principio en el capitulo 2, para la consecución de la meta, una de las herramientas utilizadas para el ambiente de desarrollo fue Eclipse IDE. Importante destacar se trata de Eclipse Java EE IDE, version Kepler, para plataformas Linux (específicamente ubuntu 12.04 LTS precise pangolin). En el caso particular de este trabajo, la arquitectura de la pc fue de 64 bits, sin embargo, esto es mencionado como referencia, ya que para máquinas de 32 bits no deberá suponer mayores inconvenientes. Antes de instalar Eclipse es necesario tener Java instalado y accesible en el PATH.

1. Descargar eclipse de: http://www.eclipse.org/
   Ver "Downloads -> Eclipse IDE for Java EE Developers", el archivo a descargar tiene un nombre similar a:
   eclipse-jee-kepler-SR1-linux-gtk-x86_64.tar.gz

   Nota: Es importante descargar la versión para desarrollo empresarial: Java Enterprise Edition (JEE)

   – Los siguientes pasos deben hacerse como super-usuario (root) –

2. Copiar o mover, usando el comando cp o mv, el archivo descargado al directorio /usr/local/src

3. Ubicarse, utilizando el comando cd (Ej: cd /usr/local/src), en el directorio /usr/local/src

4. Descomprimir el archivo descargado usando el comando tar:
   eclipse-jee-kepler-SR1-linux-gtk-x86_64.tar.gz

5. Mover el directorio generado en el paso 4 al directorio /usr/local/ por ejemplo:
   mv eclipse /usr/local/

6. Ubicarse en el directorio /usr/local (Ej: cd /usr/local)

7. Para posteriormente poder instalar plug-ins de eclipse (Ej: el cliente SVN) con el gestor de plug-ins de la herramienta, es recomendable que el usuario que normalmente usa el eclipse tenga permisos de escritura sobre el directorio donde este fue instalado, esto se puede lograr usando el comando chown desde el directorio /usr/local:
   chown usario.usuario eclipse -R
   Donde debe reemplazar usuario.usuario por el usuario.grupo correspondiente a la cuenta desde la que se utilizará eclipse, por ejemplo, si usa una cuenta "pedro" normalmente entonces será:
   chown pedro.pedro eclipse -R
   – Los siguientes pasos deben hacerse como un usuario normal (NO como root) –

8. Si entra al directorio /usr/local/eclipse y ejecuta:
   ./eclipse
   El programa debería ejecutarse correctamente.

9. Adicionalmente si desea que eclipse está en el PATH para que pueda escribir en eclipse directamente desde cualquier lugar y cualquier terminal, al igual que en el caso de Java, debe añadirlo al .bashrc o al .bash_profile. En el caso de eclipse, el .basrc o .bash_profile, quedaría aproximadamente de la siguiente forma:
   export ECLIPSE_HOME=/usr/local/eclipse
   PATH=$ECLIPSE_HOME:$PATH

5.1.3 Instalación de Apache Tomcat

Antes de instalar Apache Tomcat es necesario tener Java instalado y accesible en el PATH.

1. Descargar tomcat de: http://tomcat.apache.org/
   Ver "Downloads -> Tomcat 7.0->Binary Distributions->tar.gz", el archivo a descargar tiene un nombre similar a:
   apache-tomcat-7.0.53.tar.gz

   – Los siguientes pasos deben hacerse como super-usuario (root) –

2. Copiar o mover, usando el comando cp o mv, el archivo descargado al directorio /usr/local/src

3. Ubicarse, utilizando el comando cd (Ej: cd /usr/local/src), en el directorio /usr/local/src

4. Descomprimir el archivo descargado usando el comando tar:
   tar zxvf apache-tomcat-7.0.53.tar.gz
   Esto generará un directorio apache-tomcat-7.0.53.

5. Ubicarse en el directorio /usr/local (Ej: cd /usr/local)

6. Para poder posteriormente utilizar tomcat desde Eclipse, es recomendable que el usuario que normalmente usa el Eclipse tenga permisos de escritura sobre el directorio donde el tomcat fue instalado, esto se puede lograr usando el comando chown desde el directorio /usr/local:
   chown usuario.usuario apache-tomcat-7.0.53 -R
   Donde debe reemplazar usuario.usuario por el usuario.grupo correspondiente a la cuenta desde la que se utilizará eclipse, por ejemplo, si usa una cuenta "pedro" normalmente entonces será:
   chown pedro.pedro apache-tomcat-7.0.53 -R
   – **Los siguientes pasos deben hacerse como un usuario normal (NO como root)** –

7. Para probar que el tomcat está funcionando adecuadamente entre en el directorio /usr/local/apache-tomcat-7.0.53/bin
   y ejecute el script del arranque de tomcat:
   ./startup.sh
   Debe ver algo como:
   gustavo@gustavo: $ ./startup.sh
   Using CATALINA_BASE: /usr/local/apache-tomcat
   Using CATALINA_HOME: /usr/local/apache-tomcat
   Using CATALINA_BASE: /usr/local/apache-tomcat
   Using CATALINA_TMPDIR: /usr/local/apache-tomcat/temp
   Using JRE_HOME: /usr/local/java

   Luego ejecute un navegador y abra la dirección:
   http://127.0.0.1:8080
   Debe ver abrir una pagina como la siguiente:

8. Para detener el tomcat, debe ejecutar el script ./shutdown.sh
   gustavo@gustavo: $ ./shutdown.sh
   Using CATALINA_BASE: /usr/local/apache-tomcat
   Using CATALINA_HOME: /usr/local/apache-tomcat
   Using CATALINA_BASE: /usr/local/apache-tomcat

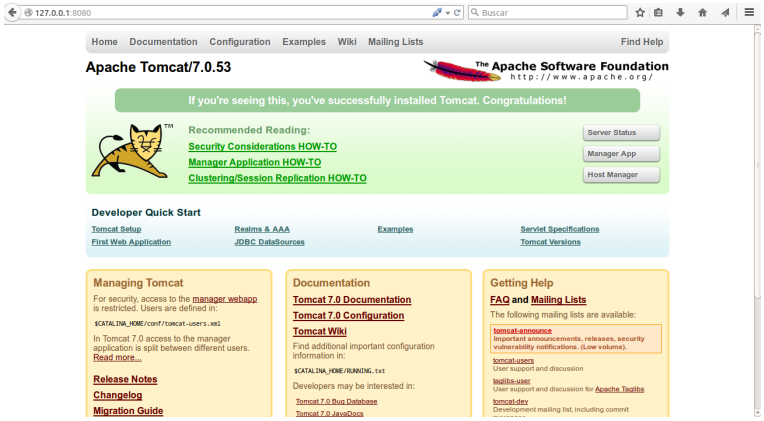

Figura 5.1: Servidor Apache Tomcat

Using CATALINA_TMPDIR: /usr/local/apache-tomcat/temp
Using JRE_HOME: /usr/local/java

5.1.4 Instalación de Axis 2

1. Se descarga la distribución binaria de Axis2 así como la distribución WAR desde http://axis.apache.org/axis2/java/core/download.cgi

2. Se descomprime el .zip axis2-1.4.1-war y se copia el archivo axis2.war en la ruta donde se publican los distintos servicios dentro de Apache Tomcat, que en este caso estaría en la ruta /usr/local/apache-tomcat-7.0.53/webapps

3. Se descomprime "axis2-1.6.2-bin.zip" en la ruta /usr/local/

4. Se establece una nueva variable de entorno denominada AXIS2_HOME y se anexa al PATH. Esto se logra anexando al archivo .bashrc el siguiente script:

```
export AXIS2_HOME=/usr/local/axis2-1.6.2
PATH=$AXIS2_HOME/bin:$PATH
```

5. Finalmente se inicia el servidor apache de manera que el .war de Axis se cargue y posteriormente en cualquier navegador web se accede a la siguiente dirección:

http://localhost:8080/axis2/

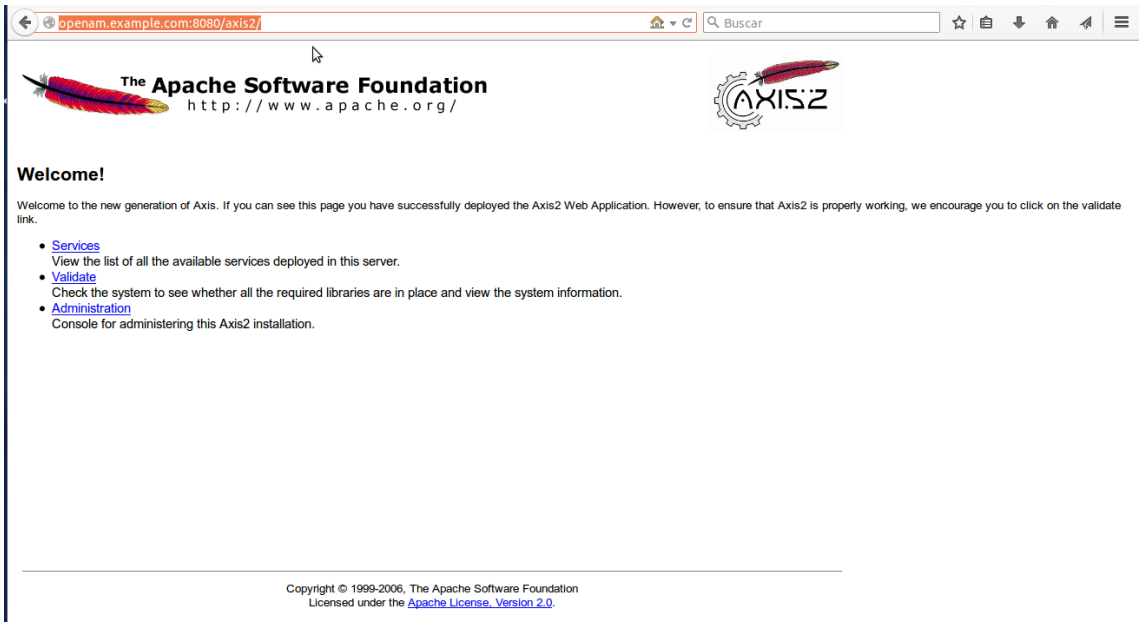

Figura 5.2: Servidor Axis2 iniciado

Nótese que en el caso de la figura, en la barra de dirección url, el texto mencionado anteriormente no aparece exactamente igual, (http://localhost:8080/axis2/), sino que refleja la dirección "http://openam.example.com:8080/axis2/". La razón de ello será explicada adelante en el siguiente apartado.

5.1.5 Instalación de OpenAM

En este ámbito se recoge los requisitos y pasos necesarios para instalar el software OpenAM, incluyendo como preparar el servidor de aplicaciones, y como correr OpenAM .

5.1.5.1 Preparación de un nombre completamente calificado para DNS

OpenAM requiere que se provea un nombre completamente calificado (FQDN: Fully Qualafied Domain Name) al momento de iniciar su configuración. Antes de

comenzar el proceso, es necesario asegurarse que el sistema tiene un FQDN tal como "openam.example.com". Para propósitos de evaluación, se puede dar al sistema un alias usando el archivo /etc/hosts en sistemas basados en UNIX. Accediendo a dicho archivo se deberá hacer una modificación al script, por ejemplo en el caso particular de este trabajo, se ve asi:

```
127.0.0.1 openam.example.com
#127.0.0.1 localhost
127.0.1.1 gustavo
```

No se debe usar el dominio localhost para el acceso de OpemAM, ni siquiera para propósitos de pruebas, porque OpenAM se basa fuertemente en cookies de navegador. Además el nombre de dominio que se escoja, debe contener al menos dos '.' puntos como caracteres, como openam.example.com. Esto le da a la instancia un dominio "example", y un subdominio "openam". Los subdominios proveen una capa extra de organización para diferentes departamentos o tipos de usuarios o clientes. También provee un nivel de control sobre los dominios de primer nivel que no se podría tener si se usara .com ó .org

5.1.5.2 Obteniendo el software

1. Descargar OpenAM desde http://forgerock.org/openam.html. Para esta entrega la cual es 11.0.0 de OpenAM con todos los servicios, se puede descargar el paquete entero , ya sea como un archivo .zip, o solamente el archivo OpenAM.war/

2. Como se mencionaba antes, el servidor contenedor que se ha usado es apache tomcat, y el entorno de desarrollo es Eclipse, por lo cual se debe hacer una pequeña modificación previa al arranque. Esto se debe a que el núcleo de OpenAM requiere que la maquina virtual (JVM) tenga un tamaño mínimo de heap de 1 Gb, y un tamaño de generación permanente de 256 Mb. Para lograr este acometido es necesario agregar una modificación a la configuración del

lanzador del tomcat. La ruta para ello es la siguiente: Open launch configuration->Arguments. Y el script es:

```
-Xmx1024m -XX:MaxPermSize=256m
```

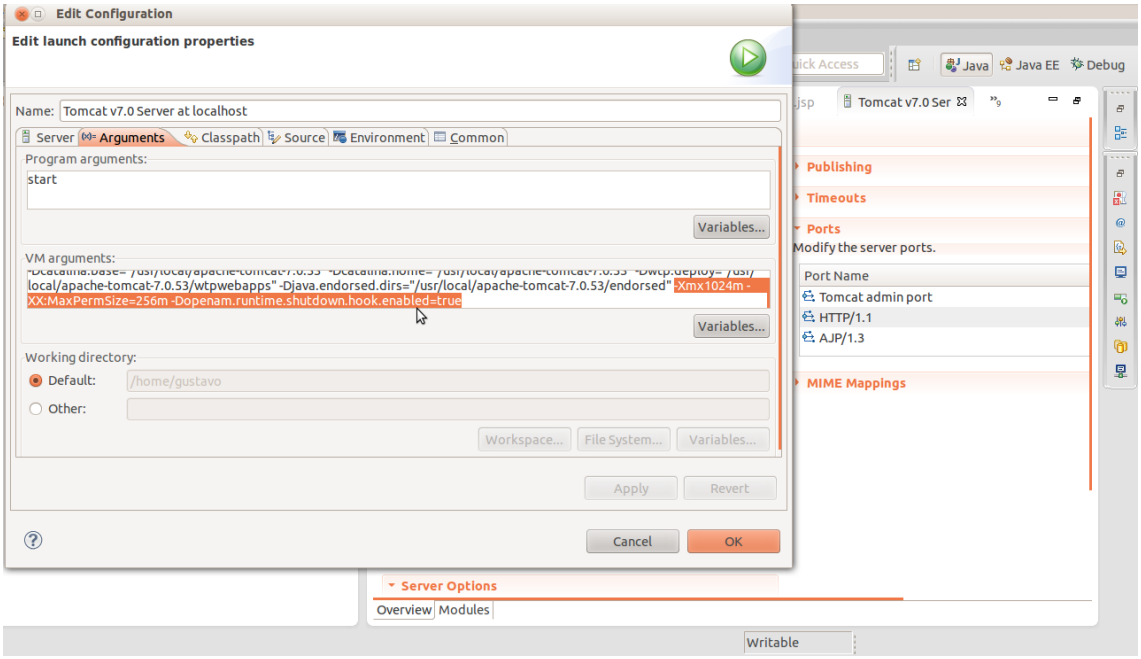

Figura 5.3: Modificando el script.

5.1.5.3 Despliegue de Tomcat

El archivo "OpenAM-11.0.0.war" contiene todos los componentes del servidor OpenAM y muestras. Como ejecutar el archivo .war depende en el contenedor de aplicaciones web.

1. Desplegar el .war en el contenedor se logra con el copiado del .war al directorio de apache tomcat a través del siguiente comando:

```
$cp Descargas/OpenAM-11.0.0.war
/usr/local/apache-tomcat-7.0.53/webapps/openam.war
```

Nota: Al agregar openam.war al final de la ruta, el nombre del archivo .war, será cambiado de OpenAM-11.0.0.war a openam.war para hacer la ejecución mas fácil.

2. Se corre el servidor tomcat desde eclipse para verificar que todo salió bien.

3. Finalmente se verifica si la pagina inicial en el navegador se muestra accediendo a:

```
openam.example.com:8080/openam
```

Y luego se obtendrá algo parecido a la figura 5.4:

5.1.6 Segmento de código de SAML

Se muestra el despliegue de un perfil de navegador web SSO donde tanto el proveedor de servicio (SP), y el proveedor de identidad (IdP) usan el enlace HTTP POST. El flujo de mensaje comienza con una petición de recurso seguro al SP.

1. **Solicita el recurso con el SP**
   El principal (vía un agente HTTP) solicita un recurso al proveedor de servicio:

```
https://sp.example.com/myresource
```

El proveedor de servicio realiza una verificación de seguridad en nombre del recurso. Si un contexto seguro valido del proveedor de servicio ya existe se omite pasos del 2-7

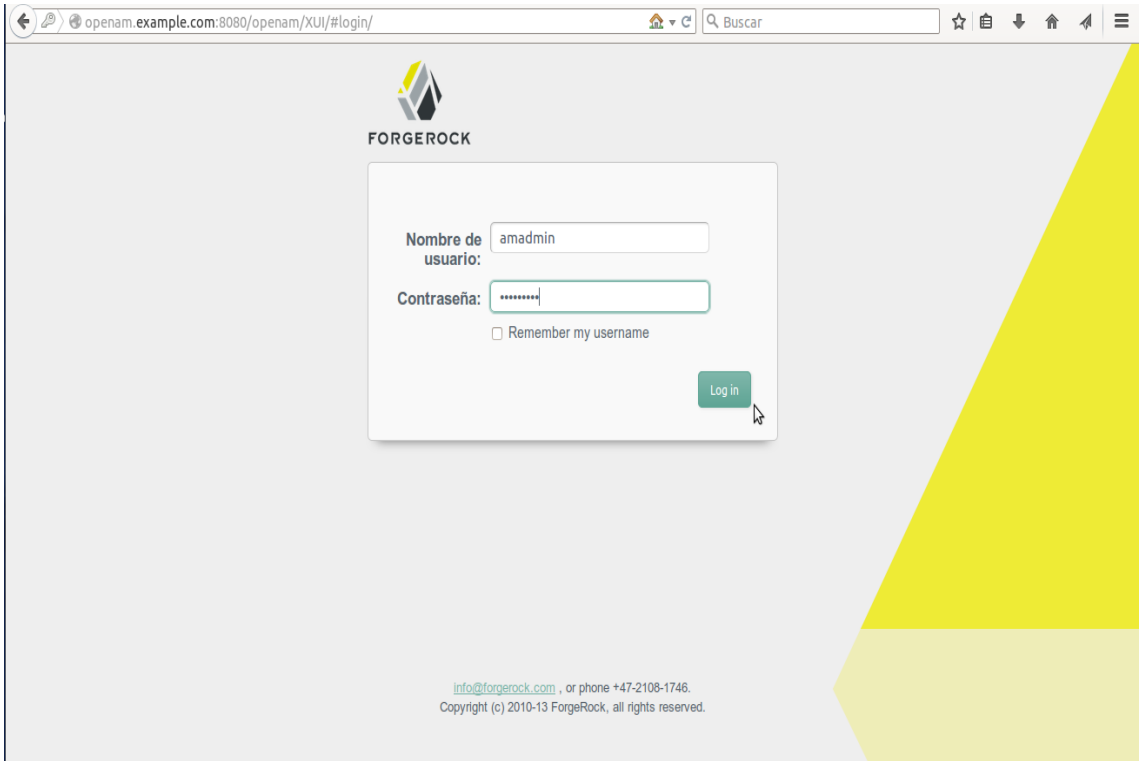

Figura 5.4: Interfaz de usuario.

2. **Respuesta con un formulario XHTML**
   El proveedor de servicio responde con un documento conteniendo un formulario XHTML.

```
<form method="post" action="https://idp.example.org/SAML2/SSO/">
  <input type="hidden" name="SAMLRequest" value="request" />
  <input type="hidden" name="RelayState" value="token" />
  ...
  <input type="submit" value="Submit" />
</form>
```

El token RelayState is un una referencia vaga a la información del estado mantenida por el proveedor de servicio. El valor del parámetro SAMLRequest es de codificacion base64 del siguiente elemento $< samlp : AuthnRequest >$:

```
<samlp:AuthnRequest
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="identifier_1"
  Version="2.0"
  IssueInstant="2004-12-05T09:21:59"
  AssertionConsumerServiceIndex="0">
  <saml:Issuer>https://sp.example.com/SAML2</saml:Issuer>
  <samlp:NameIDPolicy
    AllowCreate="true"
    Format="urn:oasis:names:tc:SAML:2.0:
    nameid-format:transient"/>
</samlp:AuthnRequest>
```

Antes de que el elemento $< samlp : AuthnRequest >$ sea insertado en el formulario XHTML, es primero codificado a base64.

3. **Solicitud de servicio SSO al IdP**
   El agente somete una petición POST al servicio SSO al proveedor de identidades:

```
POST /SAML2/SSO/POST HTTP/1.1
 Host: idp.example.org
 Content-Type: application/x-www-form-urlencoded
 Content-Length:

 SAMLRequest=request&RelayState=token
```

Donde los valores de los parámetros SAMLRequest y RelayState son tomados de el formulario XHTML en el paso 2. El servicio SSO procesa el elemento $< samlp : AuthnRequest >$ (mediante el decodificado de URL y decodificado base64 y

procesando la petición, en ese orden) y realiza una verificación de seguridad. Si el usuario no tiene un contexto seguro valido, el proveedor de identidades, identifica al usuario.

4. **Respuesta con un formulario XHTML** El servicio SSO valida la petición y responde con un documento que contiene un formulario XHTML.

```
<form method="post" action="https://sp.example.com/SAML2/SSO
/POST" ...>
   <input type="hidden" name="SAMLResponse" value="response" />
   <input type="hidden" name="RelayState" value="token" />
   ...
   <input type="submit" value="Submit" />
 </form>
```

El valor del parámetro RelayState ha sido preservado en el paso 3. El valor del parámetro SAMLResponse es de codificación base64 del siguiente elemento $< samlp : Response >$

```
<samlp:Response
    xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="identifier_2"
    InResponseTo="identifier_1"
    Version="2.0"
    IssueInstant="2004-12-05T09:22:05"
    Destination="https://sp.example.com/SAML2/SSO/POST">
    <saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
    <samlp:Status>
      <samlp:StatusCode
        Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
```

```
</samlp:Status>
<saml:Assertion
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="identifier_3"
  Version="2.0"
  IssueInstant="2004-12-05T09:22:05">
  <saml:Issuer>https://idp.example.org/SAML2</saml:Issuer>
  <!-- a POSTed assertion MUST be signed -->
  <ds:Signature
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...
    </ds:Signature>
  <saml:Subject>
    <saml:NameID
      Format="urn:oasis:names:tc:SAML:2.0:
      nameid-format:transient">
      3f7b3dcf-1674-4ecd-92c8-1544f346baf8
    </saml:NameID>
    <saml:SubjectConfirmation
      Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
      <saml:SubjectConfirmationData
        InResponseTo="identifier_1"
        Recipient="https://sp.example.com/SAML2/SSO/POST"
        NotOnOrAfter="2004-12-05T09:27:05"/>
    </saml:SubjectConfirmation>
  </saml:Subject>
  <saml:Conditions
    NotBefore="2004-12-05T09:17:05"
    NotOnOrAfter="2004-12-05T09:27:05">
    <saml:AudienceRestriction>
      <saml:Audience>https://sp.example.com/SAML2
```

```
            </saml:Audience>
          </saml:AudienceRestriction>
        </saml:Conditions>
        <saml:AuthnStatement
          AuthnInstant="2004-12-05T09:22:00"
          SessionIndex="identifier_3">
          <saml:AuthnContext>
            <saml:AuthnContextClassRef>
              urn:oasis:names:tc:SAML:2.0:ac:classes:
              PasswordProtectedTransport
           </saml:AuthnContextClassRef>
          </saml:AuthnContext>
        </saml:AuthnStatement>
      </saml:Assertion>
    </samlp:Response>
```

5. **Solicitud de la aserción del consumidor al SP**
   El agente emite una petición POST a la aserción del servicio consumidor al proveedor de servicio:

```
 POST /SAML2/SSO/POST HTTP/1.1
Host: sp.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: nnn

SAMLResponse=response&RelayState=token
```

   Donde los valores de los parámetros SAMLResponse y RelayState, son tomados del formulario XHTML del paso 4.

6. **Redirigir el recurso objetivo**
   La aserción del servicio consumidor procesa la respuesta, crea un contexto seguro en el proveedor de servicio y redirige al agente al recurso objetivo.

7. **Solicita el recurso al SP nuevamente**
   El agente solicita el recurso objetivo al proveedor de servicio nuevamente.

```
https://sp.example.com/myresource
```

8. **Respuesta con el recurso solicitado** Ya que el contexto de "seguridad" existe, el proveedor de servicio retorna el recurso para el agente.

Esto es un esquema básico y relativamente simple de lo que constituye un despliegue de SAML 2.0 del perfil SSO de un navegador Web, donde tanto el proveedor de servicios también abreviado "SP" (Service Provider) y el proveedor de identidad (Identity Provider) abreviado "IdP" usan la atadura HTTP POST.

Bibliografía

Boehm, B. (2000). *Spiral Development: Experience, Principles, and Refinements*. Spiral Development Workshop.

Icke, I. (2007). *Desarrollo en espiral*. http://es.wikipedia.org/wiki/Desarrollo_en_espiral.

Scavo, T. (2007). *SAML 2.0*. http://en.wikipedia.org/wiki/SAML_2.0#/media/File:Saml2browsersso
-post.gif.

Sokpop, S. (2009). *Sobre SOAP*. http://es.wikipedia.org/wiki/Simple_Object_Access_Protocol#/medi

Banks, J. Carson, J. (2004). *Discrete-Event System Simulation*. Prentice Hall.

Fawcett, J. Quin, L. and Ayers, D. (2012). *Beginning XML*. John Wiley and Sons, Inc.

Uzcátegui, M. and Dávila, J. and Tucci, K. (2011). *Galatea: una historia de modelado y simulación*. Revista Ciencia e Ingeniería. Edición Especial: "Jornada de Modelado y Simulación".

Rengifo, B. (2011). *Desarrollo de un servicio web para la modeloteca del sistema nacional de simulación*. Universidad de Los Andes Mérida, Venezuela.

Sommerville, I. (2005). *Ingeniería del software*. Pearson Educación, Madrid 2005 S.A.

Fensel, D. Kerrigan, and Zaremba. M (2008). *Implementing Semantic Web Services*. Springer.

Daigneau, R. (2012). *Service Design Patterns Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. Addison-Wesley.

Kalin, M. (2013). *Java Web Services: Up and Running*. O'Reilly Media, Inc.

Hewitt, E. (2009). *Java SOA Cookbook*. O'Reilly Media, Inc.

Fowler, S. and Stanwick, V. (2004). *WEB APPLICATION DESIGN HANDBOOK: Best Practices for Web-Based Software*. Elsevier.

K, Ramarao. C, Prasad. (2008). *SOA Security*. Manning Publications Co.

Deitel,P. Deitel, H (2012). *Java How to Program*. Prentice Hall.

Silberschatz, A. Korth, F. and Sudarshan, S (2011). *Database System Concepts*. McGraw-Hill.

Boehm, B. (1986). *A Spiral Model of Software Development and Enhancement*. TRW Defense Systems Group.

Craig, M. (2013). *OpenAM 11.0.0 Developer's Guide*. ForgeRock AS.

Craig, M. Richie, V. and Jang, M (2013). *OpenAM 10.1.0 Installation Guide*. ForgeRock AS.

Craig, M. (2013). *OpenAM 10.1.0 Administration Guide*. ForgeRock AS.

Langer, A. (2012). *Guide to Software Development Designing and Managing the Life Cycle*. Springer.

Hurwitz,J. Bloor, R. Baroudi,C.and M Kaufman, C. (2007). *Service Oriented Architecture For Dummies*. Wiley Publishing, Inc.

Symington, S. F. Morse, K. Petty, M. (2000). *IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA)?Object Model Template (OMT) Specification*. IEEE-SA Standards Board.

Padilla, V. (2013). *Modelo de Referencia para Geosimulación con agentes y base de datos*. Universidad Nacional Experimental de Guayana, Ciudad Guayana.

Nutaro, J. (2011). *Building Software For Simulation, Theory and Algorithms, with Applications in C++*. John Wiley & Sons, Inc.

Thangasamy, I. (2011). *OpenAM*. Packt Publishing Ltd.

Murach, J. (2012). *Murach's MySQL, Reference and Training*. Mike Murach and Associates, Inc.

Vohra, D. (2012). *Java EE Development with Eclipse*. Packt Publishing.

Dai,N. Mandel, L. and Ryman, A. (2007). *Eclipse Web Tools Platform*. Addison-Wesley.

Pressman, R. (2005). *Ingeniería del Software un enfoque practico*. McGraw-Hill.