Informe final – Trabajo de grado

Un modelo para la integración de mecanismos de autenticación de usuarios

Licenciatura en Informática

Facultad de Informática
Universidad Nacional de La Plata

Director: Bibbó, Luis Mariano
Co-Director: Fernández, Alejandro

Martinez, Fernando Carlos
fernando.martinez@sol.info.unlp.edu.ar

Miguez, Alejandro Abel
ale_miguez@hotmail.com

27 de Junio de 2005

Agradecimientos

Primero, queríamos agradecer a Casco y Luisma que nos condujeron en la senda recorrida a lo largo de este trabajo, ofreciéndonos toda su experiencia y ayuda para concluir satisfactoriamente nuestra tesis.

Agradecemos enormemente a Paola y Paula que nos cebaron mate un sinnúmero de veces mientras hacíamos este trabajo. A nuestras familias que nos presionaron y estuvieron siguiéndonos muy de cerca para que de una vez por todas llegáramos al final de la carrera.

A nuestros amigos y amigas que vienen esperando la fiesta desde hace mucho tiempo, juntando huevos y todo tipo de desperdicios, tanto orgánicos como inorgánicos, para ser depositados en nuestros delicados cuerpos.

Ale y Fer

Por haber estado conmigo, en TODO momento, a lo largo de estos últimos años, y hacer mucho más fáciles los momentos realmente complicados. Por ayudarme a ponerme las pilas cuando la verdad que no tenía muchas ganas y por alentarme a hacer lo que me parecía correcto. A vos Pau, que sos la persona más importante para mi.

A Carlini y Susi, que más allá de sus sermones y levantadas en peso, me arrastraron hasta la meta. A Andre, que siempre fue un destino seguro cuando tuve que buscar experiencia u otra perspectiva.

A Lichis, Pato, Steve, Abel, H y Rako que estuvieron siempre, en diferentes ámbitos y para diferentes cosas, pero siempre con la mejor predisposición.

Y a Ale, mi compañero de siempre, mi amigo personal, con el cual recorrimos casi el 100 % de la carrera, y que me pone muy contento y me llena de orgullo poder concluirla con él.

Fer

Quiero agredecer a mis viejos, que a pesar de todas las dificultades, hicieron un esfuerzo enorme, y me brindaron la posibilidad de venir a estudiar, y así poder forjarme un futuro.

A mis abuelos, que siempre estaban pendientes de los avances en la carrera, y me daban todo su apoyo.

A Paola, que me acompañó en los últimos años de carrera, y en quién encontre una verdadera compañera de vida, que espero que siga conmigo por muchos años más.

A Esteban y Mariana, que siempre estaban preguntando cuando había fiesta.

A todos mis compañeros de estudio, algunos de los cuales me brindaron su amistad, compartiendo salidas, partidos de truco y fulbito.

A Carlos y Susana, en los cuales encontré dos personas bárbaras, que me ayudaron y cuidaron desde que llegué a La Plata.

Y por último, a Fernando, una persona excelente, con la cual compartí toda la carrera, y me brindó su amistad desinteresadamente, y espero que sigamos siendo amigos por siempre.

Ale

Muchas gracias, a los que de una u otra manera.

Índice general

1 Introducción **4**
- 1.1 Motivación . . . 4
- 1.2 Contribuciones . . . 6
- 1.3 Organización del informe . . . 6

2 Análisis del Problema **8**
- 2.1 Seguridad . . . 9
  - 2.1.1 Autenticación . . . 9
  - 2.1.2 Autorización . . . 9
- 2.2 Administración . . . 9
- 2.3 Integración de aplicaciones . . . 10
- 2.4 Modelo . . . 11

3 Estado del Arte **13**
- 3.1 CAS (Central Authentication System) . . . 13
- 3.2 JOSSO (Java Open Single Sign-On) . . . 15
- 3.3 PubCookie . . . 17
- 3.4 Comparación entre las aplicaciones . . . 18

4 Solución Propuesta **20**
- 4.1 Servicio de autenticación/autorización . . . 22
- 4.2 Módulo para el manejo de sesión única . . . 29
  - 4.2.1 Sesión de aplicación . . . 29
  - 4.2.2 Sesión de usuario . . . 31
- 4.3 Herramienta de Administración . . . 32
- 4.4 Librerías de integración . . . 35
- 4.5 Comunicación . . . 38
- 4.6 Módulo Externo de Autenticación . . . 39

5 Implementación **41**
- 5.1 Decisiones de diseño . . . 41

5.1.1 Arquitectura 41
5.1.2 Mecanismo de Autenticación 42
5.1.3 Conceptos de **Sesión** y **Ticket** 42
5.1.4 Seguridad de las Contraseñas 43
5.2 Tecnologías 43
5.3 Implementación del modelo **IAM** 46
5.3.1 Inicio y configuración del modelo **IAM** 46
5.3.2 Servicio de autenticación/autorización **IAM** 47
5.3.3 Módulo para el manejo de sesión única 48
5.3.4 Librerías de Integración 49
5.3.5 Herramienta de Administración 50
5.3.6 Seguridad en la Comunicación 51
5.3.7 API integración módulo externo 54

6 Experiencias de Uso **60**
6.1 ChikiWiki 61
6.2 Mantis Bugtracker 62
6.3 Jabber 63

7 Conclusiones **67**
7.1 Trabajo Futuro 68

Capítulo 1

Introducción

1.1 Motivación

El ingreso a la nueva era de e-business ha traído consigo cambios significativos en la forma de trabajar e interactuar entre personas y compañías. El surgimiento de nuevas tecnologías repletas de nuevos términos y conceptos, hace que estar actualizado sea un verdadero reto para cualquier profesional de hoy en día.

El concepto de "Groupware" [1] (o también llamado "Software para Trabajo Colaborativo") es la convergencia de lo que en años anteriores se consideraban tecnologías independientes como el servicio de mensajería instantánea (IM), los workspaces colaborativos (e.g. BSCW), las hipermedias editables (e.g. wiki-wiki), servidores para el manejo concurrente de código fuente (e.g. CVS), etc.

Las características fundamentales del trabajo colaborativo son: la **comunicación**, la **coordinación** y la **colaboración**. El Groupware es entonces una herramienta que soporta las actividades nombradas anteriormente, ayudando a los individuos a trabajar juntos en un modo cualitativamente mejor.

En la práctica, una plataforma para ayudar a un equipo a desarrollar su tarea se compone de más de una de estas herramientas. Un problema difícil de resolver al integrar las distintas aplicaciones, es el manejo de usuarios.

Imagine el siguiente escenario: una organización de desarrollo de software que se encuentra distribuida en varios países decide contar con capacidades para el soporte de trabajo colaborativo; en consecuencia, provee herramientas que permiten la comunicación, coordinación y colaboración entre los distintos integrantes.

La comunicación básica [2] de los empleados de cualquier organización, está siempre relacionada con la comunicación "instantánea". Es decir, que los

diferentes empleados suelen necesitar interacción inmediata con sus colegas, ya sea para pedirle asistencia en algún aspecto de su trabajo, o para dar a conocer alguna resolución de último momento. Si bien parecería algo trivial solucionar este tipo de problemas en un ambiente donde todos los empleados conviven dentro del mismo espacio físico de trabajo, deja de serlo cuando dicho espacio se encuentra distribuido en distintas partes del mundo. Es por esto, que una herramienta de Mensajería Instantánea (IM) [3] sería clave para sortear esta dificultad de comunicación.

La interacción no solamente se da de forma inmediata. Esto quiere decir, que en muchos casos, se necesita que los distintos integrantes de una organización aporten sus conocimientos generando así un repositorio centralizado de información. Para esto, sería poco práctico tener a una persona que se encargue de recopilar lo que cada uno de los empleados puede llegar a aportar como conocimiento relevante. Es por esto que muchas organizaciones utilizan las Hipermedias Editables (Wikis) [4], que proveen un repositorio centralizado donde los diferentes empleados incorporan sus conocimientos y no se necesita de una persona que organice y genere dicha información. Una ventaja de este tipo de herramientas, es que provee una forma sencilla y estructurada de ingresar la información, sin necesidad de que el usuario posea algún conocimiento previo de HTML. Luego, el resto de los empleados, puede acceder a esta Wiki y ver qué es lo nuevo que algún colega aportó.

Si bien la comunicación es importante dentro de cualquier organización, si se mira específicamente dentro de una organización de desarrollo de software, se encontrará que un ítem clave a la hora de poner en producción un sistema, es el seguimiento de errores. Es por esto, que se han desarrollado una serie de sistemas llamados "Bug Trackers" [5]. Estos sistemas permiten llevar un registro de los distintos errores que necesitan ser corregidos, así como también de la identidad del tester que encontró el error y la persona designada para corregirlo.

Todos los sistemas anteriores deben ser protegidos contra accesos no autorizados. Para esto, presentan un mecanismo/esquema de autenticación independiente. Si bien los mecanismos son independientes, la mayoría de los mismos reúnen características similares:

- El proceso de autenticación/autorización requiere un nombre de usuario y una contraseña.
- Cada una de las aplicaciones tiene un esquema de almacenamiento de información propio para guardar la información necesaria tanto para el proceso de autenticación/autorización como para el propio funcionamiento de la misma.

Al utilizar diversas aplicaciones con las características mencionadas anteriormente, surgen una serie de problemas. Uno de estos es la necesidad de recordar múltiples usuarios y contraseñas para cada una de las aplicaciones que se están utilizando. Para no caer en este caso, se podría usar el mismo usuario y contraseña en todas las aplicaciones, pero ante un eventual cambio de contraseña, el mismo deberá hacerse conjuntamente en todas las aplicaciones para mantener la consistencia. Esto introduce una carga innecesaria para el administrador, el cual será el encargado del mantenimiento de la información en las diferentes aplicaciones. Otro problema es la necesidad de que se ingresen los usuarios y contraseñas en cada una de las aplicaciones, lo cual puede resultar tedioso, y más aún cuando se tiene el mismo nombre de usuario y contraseña para todas las aplicaciones. Además se presenta la dificultad, tanto del lado de la persona como del lado del administrador, a la hora de agregar una nueva aplicación, ya que se debe administrar la información de los usuarios de esta nueva aplicación y la persona necesita recordar más nombres de usuario y contraseñas. Es importante destacar que no todos los usuarios tienen permiso para utilizar todas las aplicaciones, sino que sólo tienen acceso a aquellas aplicaciones que les permiten desempeñar sus funciones dentro de la organización. La administración de los derechos de acceso, resulta también una carga para el administrador.

1.2 Contribuciones

Las contribuciones de este trabajo son:

1. Un modelo de autenticación/autorización único que permita integrar múltiples aplicaciones basadas en un mecanismo de autenticación que requiere nombre de usuario/contraseña.

2. Implementación de una solución de software basada en el modelo propuesto.

3. Demostración de la utilización de la solución, en un prototipo que integra un servicio de mensajería instantánea (Jabber), una hipermedia editable (ChikiWiki) y un sistema para el reporte y seguimiento de bugs (Mantis).

1.3 Organización del informe

Este informe se organiza de la siguiente forma:

En el *Capítulo 2* se presenta el análisis del problema, necesario para comprender el contexto en el cual se ubica el modelo y se presentan los requerimientos a satisfacer.

En el *Capítulo 3* se muestran las diferentes aplicaciones existentes en la actualidad, que satisfacen en forma parcial los requerimientos mencionados en el *Capítulo 2*.

En el *Capítulo 4* se presenta la arquitectura del modelo propuesto, viendo en detalle sus componentes, interacción entre ellos y funcionalidad.

En el *Capítulo 5* se describen todas las tecnologías estudiadas en la etapa de análisis para utilizar durante la etapa de implementación, y se exponen los motivos por los cuales se optó por cada una de las tecnologías seleccionadas.

En el *Capítulo 6* se muestra el correcto funcionamiento del modelo instanciándolo en un escenario determinado, mostrando las diferentes experiencias de uso dentro de dicho escenario.

En el *Capítulo 7* se detallan las conclusiones obtenidas a partir de la solución planteada y su instanciación dentro del escenario propuesto. Se hace una comparación entre las expectativas y los resultados obtenidos. También se presentan posibles trabajos futuros que se pueden realizar sobre el modelo.

Capítulo 2

Análisis del Problema

Las aplicaciones de Groupware nombradas en el *Capítulo 1*, así como también algunas otras que pueden formar parte del conjunto de aplicaciones pertenecientes a la organización, deben ser protegidas contra accesos no autorizados, y para esto, cada una presenta su propio algoritmo de autenticación, el cual es implementado de manera propietaria. Estos algoritmos pueden requerir diferente información para llevar a cabo la autenticación, así como también almacenar de distinta manera la información utilizada para este proceso.

Por otra parte, siempre existen administradores de aplicaciones que son los encargados de mantenerlas funcionales, gestionar la creación y eliminación de los distintos usuarios, y autorizar el uso de las mismas. Para realizar estas tareas, tiene que conocer en detalle la configuración de cada una de las aplicaciones. A medida que crece el número de aplicaciones, la tarea del administrador se vuelve más costosa, y por lo tanto puede llegar a perder productividad por algo que poco tiene que ver con los intereses de la organización. Si bien el administrador podría adoptar la política de utilizar siempre los mismos nombres de usuario y contraseñas para todas las aplicaciones, la información estaría almacenada directamente en cada una, y en el caso de que aparezcan nuevos usuarios o haya usuarios que ya no hagan más uso de las mismas, el administrador deberá efectuar los cambios necesarios en cada una de ellas, lo cual puede introducir algún error que conlleve a información inconsistente y por ende a accesos no autorizados a alguna de las aplicaciones. Otra causa de accesos no autorizados puede darse cuando, ante la dificultad de la persona de recordar múltiples usuarios y contraseñas, almacena esta información en algún medio externo, el cual puede quedar expuesto a algún otro usuario malintencionado.

Cuando se tienen que utilizar diversas aplicaciones, se deberá ingresar la información requerida para la autenticación en cada una de ellas, situación que resulta tediosa y poco práctica. Asimismo, la persona deberá tener pre-

sente dicha información para no sufrir retrasos al intentar acceder a alguna de ellas.

A continuación, se verán en mayor detalle algunos conceptos relacionados con esta problemática.

2.1 Seguridad

La seguridad generaliza dos características: *Autenticación* y *Autorización*.

2.1.1 Autenticación

La autenticación se refiere al hecho de determinar si una entidad es realmente quien dice ser.

En el escenario expuesto, la información que permite determinar la autenticidad es el nombre de usuario y su contraseña.

El proceso de autenticación es el siguiente:

- El usuario intenta acceder a una aplicación determinada.
- La misma le proveerá una manera de ingresar la información.
- El mecanismo de autenticación determinará si la información ingresada se corresponde con la almacenada.
- En caso de que el proceso resulte exitoso, el usuario accede a la aplicación; en caso contrario el acceso no se efectúa.

En determinadas aplicaciones, puede darse un bloqueo del usuario ante reiterados intentos fallidos, así como también la posibilidad de recordar la contraseña para futuros ingresos.

2.1.2 Autorización

La autorización determina si una entidad válida tiene permiso para efectuar la operación solicitada.

En el escenario mencionado, este proceso se basa en determinar si un usuario válido está habilitado para utilizar la aplicación solicitada.

2.2 Administración

Las aplicaciones generalmente proveen funcionalidad para administrar la información relativa a los usuarios, como por ejemplo:

- Creación, modificación y eliminación de usuarios.
- Modificación de los datos personales.
- Recuperación de contraseña.

En el escenario expuesto, en donde se integran diversas aplicaciones, se debe proveer una forma de administrar esta integración, permitiendo agregar y eliminar aplicaciones, así como también gestionar los derechos de acceso de los distintos usuarios sobre las mismas.

2.3 Integración de aplicaciones

Cada aplicación tiene su propio repositorio y mecanismo de autenticación. De esta forma, cada usuario tiene que ingresar la información de acceso para la aplicación determinada, y el administrador debe manejar individualmente la información de cada aplicación. La integración de aplicaciones permite unificar estos mecanismos, posibilitando centralizar la lógica de autenticación/autorización y la información relativa a los usuarios. Entonces, los usuarios podrán acceder a todas las aplicaciones ingresando la misma información, y el administrador tendrá que manejar menos información y de forma centralizada.

En el escenario mencionado, hay dos tipos de aplicaciones: aplicaciones *Web* y aplicaciones *Cliente/Servidor*.

Las aplicaciones *Web* son sistemas abiertos, independientes de la arquitectura de red (Internet, Intranet, etc.) y la plataforma (Windows, Unix, BSD, etc.), que solamente necesitan de algún navegador para poder accederse, sin la necesidad de instalar ningún componente de software adicional. En este tipo de aplicaciones, la comunicación es unidireccional y asincrónica. Son sistemas responsivos, es decir que la aplicación responde ante los pedidos, sin tener conocimiento de quién los realiza. Las aplicaciones *Web* necesitan de otro componente, denominado "Servidor Web", que es el encargado de alojarlas y proveer el soporte para la comunicación entre estas y los clientes.

En las aplicaciones *Cliente/Servidor* se establece una comunicación punto a punto, bidireccional y sincrónica, y tanto el cliente como el servidor conocen de la existencia del otro, pudiendo así comunicarse entre sí. Dado que la comunicación es sincrónica, un cliente realiza un pedido al servidor, y se queda esperando hasta obtener una respuesta. A diferencia de las aplicaciones *Web*, tanto los clientes como el servidor son aplicaciones autónomas, que determinan cómo y de qué forma se realizará la comunicación entre ellas.

2.4 Modelo

Para resolver adecuadamente estos problemas, se necesita un modelo que permita la integración de los dos tipos de aplicaciones mencionadas en la *Sección 2.3*, que brinde un esquema de seguridad (autenticación y autorización) único, y que provea una forma de administrar la información de manera centralizada.

Para poder integrar diferentes aplicaciones, se debe centralizar la información requerida por el esquema de seguridad en algún repositorio global. Más en detalle, se debe guardar la información necesaria para llevar a cabo tanto la autenticación (usuario y contraseña) como la autorización (permisos sobre las aplicaciones). Esta integración, debe ser lo suficientemente transparente y flexible de forma tal que el costo sea mínimo, independientemente de la arquitectura de las aplicaciones a integrar, siempre y cuando estas cumplan las características vistas en el escenario.

Toda la información utilizada por el esquema de seguridad, debe poder administrarse de forma ágil y confiable, aliviando así la carga del administrador de las aplicaciones.

Otra característica que debe satisfacer el modelo, es la de poder establecer una "sesión única" de forma tal que no haya necesidad de ingresar una y otra vez el nombre de usuario y contraseña para cada una de las aplicaciones a utilizar. Es decir, que cuando se ingrese el nombre de usuario y contraseña, se puedan utilizar todas las aplicaciones integradas sin volver a proveer esta información.

De lo analizado anteriormente, surgen los siguientes requerimientos:

- **Fácil integración**: incorporar nuevas aplicaciones de la manera más transparente posible.
- **Mecanismo de autenticación múltiple**: el modelo debe proveer un mecanismo para validar el acceso a las aplicaciones descriptas en el escenario.
- **Mecanismo de autorización**: permitir establecer permisos para utilizar las aplicaciones.
- **Independencia de la arquitectura**: el modelo debe abstraerse de las plataformas y tecnologías utilizadas por cada una de las aplicaciones.
- **Manejo de sesión única**: brindar la posibilidad de única autenticación y múltiple uso de aplicaciones.

- **Centralización de la información**: permitir almacenar la información necesaria para la autenticación y autorización en un único repositorio.
- **Administración de la información**: proveer una herramienta para gestionar la creación, modificación y eliminación de usuarios así como también de permisos sobre las aplicaciones.

Capítulo 3

Estado del Arte

En esta sección se muestran las aplicaciones actualmente utilizadas para resolver la problemática planteada en el *Capítulo 2*, explicando brevemente su funcionamiento y cómo satisfacen los requerimientos mencionados en el mismo capítulo.

Al final de la sección, se muestra una tabla comparativa de las aplicaciones, viendo los requerimientos que satisface cada una de ellas.

3.1 CAS (Central Authentication System)

CAS [6] es una aplicación open-source, y provee un mecanismo de autenticación único, centralizado, que permite integrar distintos servicios con la única restricción de que sus "front-ends" sean Web. De esta forma, brinda a los usuarios la facilidad de manejar un único nombre de usuario y contraseña para acceder a todos los servicios, y permite manejar de forma flexible los cambios en la lógica de autenticación sin tener que cambiar los servicios integrados.

Es una aplicación perteneciente a la Universidad de Yale. Está implementada como una aplicación Web stand-alone; son una serie de servlets que corren sobre un canal seguro. La arquitectura de CAS, se ve en la Figura Fig. 3.1

En cuanto a su funcionamiento, CAS se comporta de la siguiente forma: el usuario intenta acceder a una aplicación a través de una determinada URL. Este pedido es interceptado por un módulo de CAS en el Portal Application Server, y en el caso de no haber establecido previamente una sesión, se le presenta al usuario un diálogo para que ingrese su nombre de usuario y contraseña. CAS valida la información del usuario a través del Application/Service, y en caso exitoso, se establece una sesión para que en pedidos

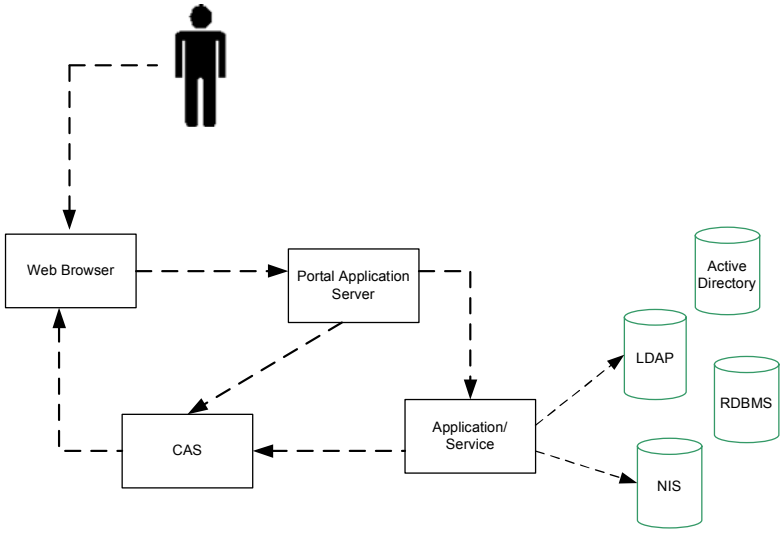

Figura 3.1: Diagrama de arquitectura de CAS

sucesivos a otros servicios, no se requiera ingresar dicha información nuevamente.

A continuación se evalúa CAS en relación a los requerimientos obtenidos del análisis del problema.

- **Fácil integración**: CAS permite integrar de forma relativamente sencilla aplicaciones, teniendo como única restricción que sus "front-ends" sean Web. Dada la arquitectura de CAS, esta integración se logra configurando los servidores de aplicaciones con las librerías correspondientes para cada caso, dependiendo del lenguaje en el cual están implementadas.

- **Mecanismo de autenticación múltiple**: CAS permite validar la información de acceso de todas las aplicaciones con un único mecanismo, permitiendo cambiar la lógica de autenticación de forma transparente para el usuario.

- **Mecanismo de autorización**: CAS no provee un mecanismo de autorización, dado que todos los usuarios de las aplicaciones integradas tendrán acceso a ellas. Esto no permite trabajar con perfiles de usuario, en los cuales se determina a qué aplicaciones tiene permiso cada uno de los usuarios.

- **Independencia de las arquitecturas**: CAS sólo soporta aplicaciones que utilizan una arquitectura Web.

- **Manejo de sesión única**: CAS maneja las sesiones mediante Cookies, que es un mecanismo estándar de los navegadores para mantener estado de la sesión del usuario.
- **Centralización de la información**: CAS provee un mecanismo abstracto de autenticación, el cual puede utilizar diferentes fuentes de información, como por ejemplo, LDAP, Active Directory o NIS Database.
- **Administración de la información**: CAS no provee una herramienta para administrar la información de autenticación. Esta administración sólo es posible a través de aplicativos propietarios del repositorio elegido.

3.2 JOSSO (Java Open Single Sign-On)

JOSSO [7] es una infraestructura open source J2EE, diseñada para proveer una plataforma neutral y centralizada de autenticación de usuarios. JOSSO provee un mecanismo de único login, transparente para los usuarios, a múltiples aplicaciones. Brinda una solución fácil de integrar con Jakarta Tomcat, proveyendo identificación de usuarios para aplicaciones Web usando el estándar Servlet Security API. Además permite implementar y combinar múltiples esquemas de autenticación con un repositorio de credenciales.

Esta desarrollada con Web Services para proveer un mecanismo de identificación de usuarios para aplicaciones Web usando el estándar Servlet Security API.

A continuación se explica el funcionamiento general de JOSSO: el usuario accede a un recurso protegido (aplicación integrada); el SSO Agent que protege la aplicación, intercepta el pedido, y dado que el usuario no está identificado, lo redirige a un formulario del SSO Gateway. El usuario ingresa aquí su credencial, que puede ser un nombre de usuario y contraseña o un certificado X.509 (dependiendo del esquema de autenticación configurado) y el SSO Gateway verifica su validez. Si la credencial es válida, el usuario autenticado y el token de sesión generado, son guardados en el medio de almacenamiento configurado. El usuario es redirigido a la aplicación originalmente solicitada, y el SSO Agent que protege la aplicación, intercepta nuevamente este pedido, y utilizando el módulo SSO Gateway JAAS, chequea la validez de la sesión y obtiene el usuario autenticado desde SSO Gateway usando SOAP. El SSO Gateway obtiene el identificador de sesión desde el repositorio de sesiones (session store) y el usuario autenticado asociado desde el repositorio de identidades (identity store), el cual es agregado y manejado por SSO Agent para la aplicación web solicitada.

En la figura Fig. 3.2 se muestra un diagrama de la arquitectura de JOSSO, componentes y relaciones entre ellos.

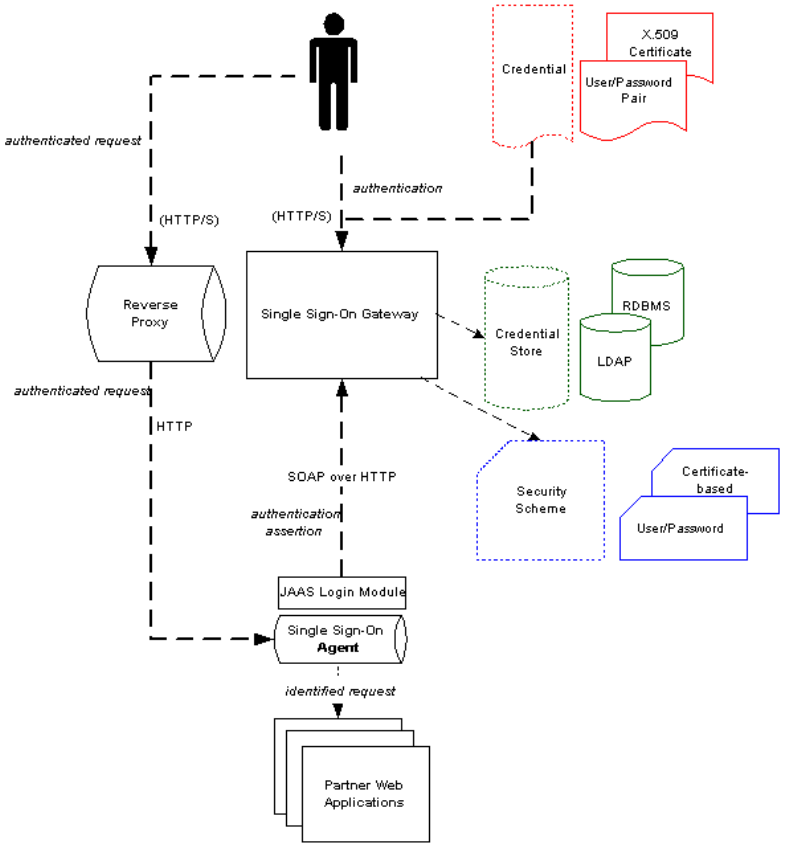

Figura 3.2: Diagrama de arquitectura de JOSSO

A continuación se evalúa JOSSO en relación a los requerimientos obtenidos del análisis del problema.

- **Fácil integración**: JOSSO permite integrar aplicaciones Web (Java, PHP, ASP, etc), usando el protocolo SOAP. Provee una solución fácil de integrar con Jakarta Tomcat.
- **Mecanismo de autenticación múltiple**: JOSSO provee un mecanismo de único login, a múltiples aplicaciones, transparente para los usuarios.
- **Mecanismo de autorización**: JOSSO no provee un mecanismo de autorización, dado que, al igual que en CAS, los usuarios tienen acceso a todas las aplicaciones integradas.

- **Independencia de las arquitecturas**: JOSSO sólo permite la integración de aplicaciones que utilizan una arquitectura Web.
- **Manejo de sesión única**: JOSSO utiliza Cookies para mantener la información de sesión del usuario, evitando de este modo, ingresar nombre de usuario y contraseña cada vez que se quiera acceder a una aplicación.
- **Centralización de la información**: JOSSO implementa un Pluggable Framework para permitir la implementación de múltiples esquemas de autenticación y almacenamiento (LDAP, XML).
- **Administración de la información**: JOSSO no provee una herramienta para la administración de la información.

3.3 PubCookie

PubCookie [8] es un modelo de autenticación de usuarios basado en cuatro componentes principales: User Agent, Application Server, Login Server y Authentication Service.

La distribución de PubCookie consta de una implementación de Login Server (programa CGI desarrollado en C), distintos Authentication Service desarrollados en C, para integrar con LDAP, Kerberos, y módulos de configuración del Application Server (Apache).

El funcionamiento de PubCookie es el siguiente: el usuario hace un requerimiento al Applicaction Server mediante el User Agent, ingresando una URL determinada. El módulo de PubCookie intercepta este pedido y verifica si ya existe una sesión establecida para el usuario. Si se encuentra disponible la información de sesión, el usuario accede a la aplicación solicitada. En caso contrario, se le presenta al usuario un diálogo para ingresar el nombre de usuario y contraseña, los cuales son validados por el Login Server a través del Authentication Service. Si la información provista por el usuario es correcta, se establece la sesión, permitiendo de esta forma, acceder a las demás aplicaciones integradas sin necesidad de reingresar esta información.

En la figura Fig. 3.3 se muestra la arquitectura de PubCookie.

A continuación se evalúa el comportamiento de PubCookie en relación a los requerimientos ya mencionados.

- **Fácil integración**: PubCookie permite integrar únicamente aplicaciones Web.
- **Mecanismo de autenticación múltiple**: PubCookie provee un mecanismo de único login, transparente para los usuarios.

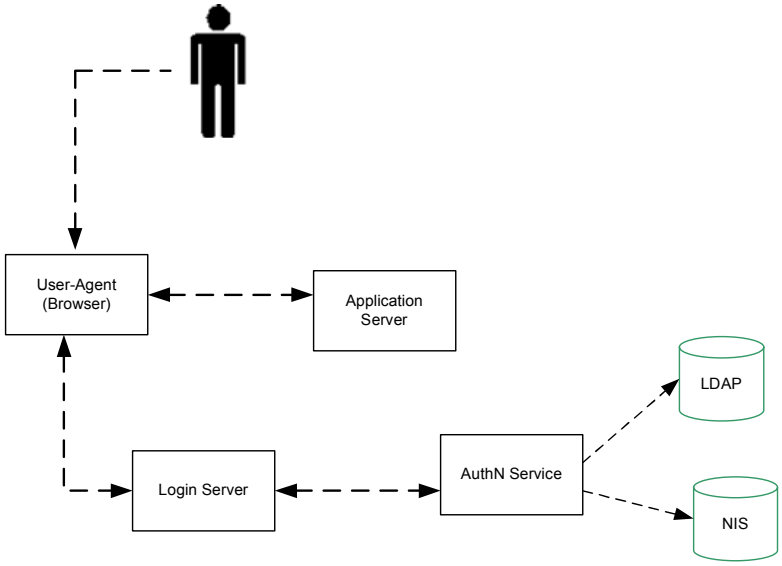

Figura 3.3: Diagrama de arquitectura de PubCookie

- **Mecanismo de autorización**: PubCookie no provee un mecanismo de autorización, dado que al igual que en CAS y JOSSO, todas las aplicaciones integradas son accesibles por los usuarios.
- **Independencia de las arquitecturas**: PubCookie sólo soporta aplicaciones desarrolladas sobre una arquitectura Web.
- **Manejo de sesión única**: PubCookie maneja la información de sesiones utilizando Cookies, las cuales son un mecanismo dependiente del navegador utilizado.
- **Centralización de la información**: PubCookie permite configurar un servicio externo de autenticación (Kerberos, LDAP, NIS), el cual tiene un repositorio de información propio.
- **Administración de la información**: PubCookie no provee ninguna herramienta para administrar la información de autenticación.

3.4 Comparación entre las aplicaciones

En la Fig. 3.4 se muestra una tabla comparativa entre las tres aplicaciones ya vistas, en relación a los requerimientos planteados en el *Capítulo 2*.

	CAS	JOSSO	PubCookie
Fácil integración	✓	✓	✓
Mecanismo de autenticación múltiple	✓	✓	✓
Mecanismo de autorización			
Independencia de las arquitecturas			
Manejo de sesión única	✓	✓	✓
Centralización de la información	✓	✓	✓
Administración de la información			

Figura 3.4: Tabla comparativa de las aplicaciones vistas

Capítulo 4

Solución Propuesta

La integración de los distintos servicios de Groupware, como los workspaces colaborativos, hipermedias editables, servidores para el manejo concurrente de código y servicios de mensajería instantánea, presenta varias dificultades: la administración de la información, la necesidad de recordar varios nombres de usuario y contraseñas para poder acceder a las mismos e ingresar esta información una y otra vez en cada uno de ellos.

De este modo surgen una serie de requerimientos, presentados en el *Capítulo 2*, que se deben satisfacer: fácil integración, abstracción de la plataforma y arquitectura de las aplicaciones, mecanismo de autenticación único, administración de la información de forma ágil y confiable, manejo de sesión única.

Esta tesis propone un modelo que brinda una solución a los requerimientos vistos anteriormente. El modelo presenta ciertos componentes, los cuales se detallan a continuación.

1. **Servicio de autenticación/autorización**: este es el componente principal del modelo, el cual permite tener un único mecanismo de autenticación y autorización a varias aplicaciones. Utiliza un protocolo de comunicación estándar e independiente de las arquitecturas y plataformas, posibilitando así una integración dinámica con diversos esquemas de seguridad. Este servicio describe de forma clara cómo debe ser la comunicación con las aplicaciones de Groupware a integrar, como así también, la comunicación con el/los módulo/s externo/s que proveen la información necesaria para la autenticación y autorización de los usuarios de las distintas aplicaciones.

2. **Módulo para el manejo de sesión única**: es una parte opcional del modelo, el cual permite a los usuarios establecer, la primera vez que acceden a una aplicación integrada, una sesión única de usuario, de

forma tal que no se requiera ingresar nuevamente el nombre de usuario y la contraseña para utilizar las demás aplicaciones. En caso de no utilizar este módulo, se establecen sesiones de aplicación.

3. **Herramienta de Administración**: es una aplicación que permite manejar la información necesaria para el proceso de autenticación y autorización. Provee básicamente la administración de usuarios, aplicaciones y permisos sobre las mismas. De esta forma se le facilita la tarea al administrador de las aplicaciones integradas, el cual podrá, de forma centralizada, manejar la información de todos los usuarios y aplicaciones, pudiendo así integrar nuevas aplicaciones y usuarios con un impacto mínimo.

4. **Librerías de integración**: el modelo provee librerías en los lenguajes más populares, las cuales permiten que se integren nuevas aplicaciones de una forma ágil y sencilla. Estas librerías, definen un módulo cliente, el cual se comunica con el servicio para realizar el proceso de autenticación/autorización, habilitando al usuario el acceso según corresponda.

Estos cuatro componentes son los que constituyen el modelo de autenticación y autorización de usuarios **IAM** (Integrated Authentication Model), el cual contempla y resuelve los requerimientos antes mencionados. La arquitectura del modelo se ve en la Fig. 4.1

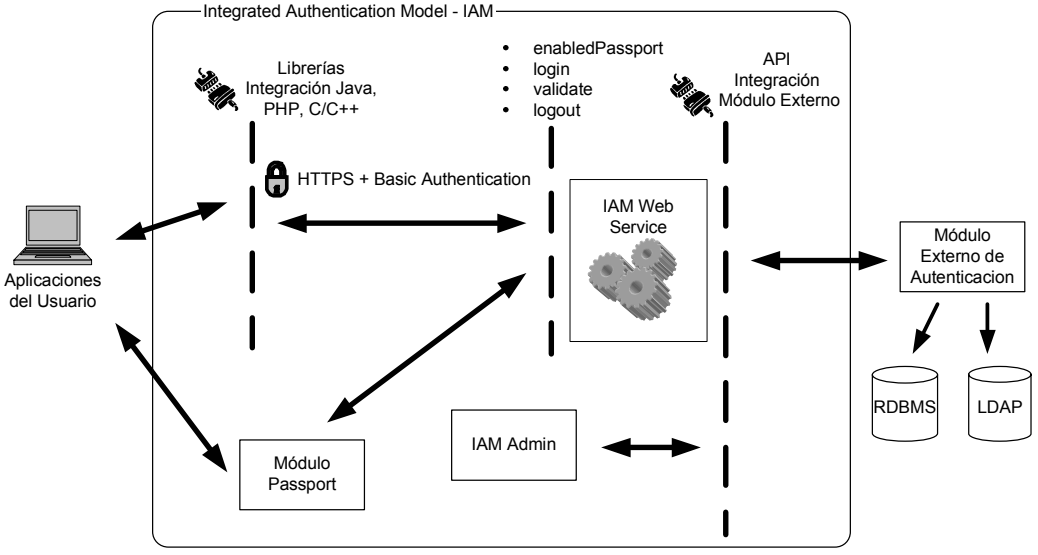

Figura 4.1: Diagrama de arquitectura del modelo **IAM**

Como se puede apreciar en la Fig. 4.1, además de los cuatro componentes del modelo **IAM**, se encuentra el módulo externo de autenticación, el cual provee toda la información necesaria para completar el funcionamiento del mecanismo de autenticación/autorización. Este módulo se tiene que adaptar al modelo según ciertas reglas de integración, permitiendo la comunicación de este con el servicio de autenticación/autorización **IAM** y la herramienta de administración de información **IAM Admin**.

El resto de esta sección se organiza de la siguiente manera:

En la *Sección 4.1* se explica con mayor detalle el servicio de autenticación/autorización, mostrando su arquitectura, funcionalidad e interacción con los demás componentes del modelo.

En la *Sección 4.2* se explica el funcionamiento y se muestran las características del módulo de sesión única.

En la *Sección 4.3* se muestra la herramienta de administración de información **IAM Admin**, características, y funcionalidades implementadas.

En la *Sección 4.4* se mencionan las distintas librerías de integración provistas, para facilitar la comunicación de las aplicaciones de usuario con el modelo **IAM**.

En la *Sección 4.5* se explica cómo es el mecanismo de comunicación entre los distintos componentes del modelo, viendo ventajas y desventajas de su uso.

En la *Sección 4.6* se explica la integración del modelo con el módulo externo de autenticación y se dan algunas recomendaciones de uso.

4.1 Servicio de autenticación/autorización

El servicio de autenticación/autorización **IAM** es el componente principal del modelo. Permite tener un único mecanismo de autenticación y autorización a varias aplicaciones, utiliza un protocolo de comunicación estándar e independiente de las arquitecturas y plataformas, y posibilita una integración dinámica con diversos esquemas de seguridad.

Se implementa a través de un Servicio Web que puede estar disponible sobre Internet/Intranet, utiliza un sistema estándar de envío de mensajes XML,

y no está ligado a ningún Sistema Operativo ni lenguaje de programación, como se ve en la Fig. 4.2.

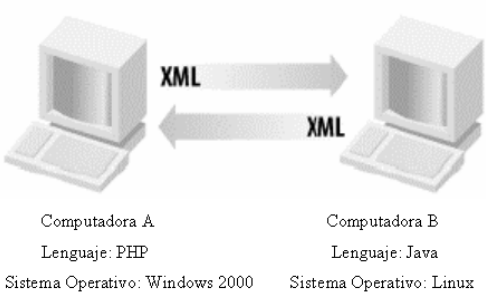

Figura 4.2: Servicio Web básico

Existen varias alternativas para el envío de mensajes XML: se puede usar XML Remote Procedure Calls (XML-RPC), SOAP (Simple Object Access Protocol), o transmitir arbitrariamente documentos XML a través de HTTP GET/POST. Estos tres tipos de comunicación se ven en la Fig. 4.3

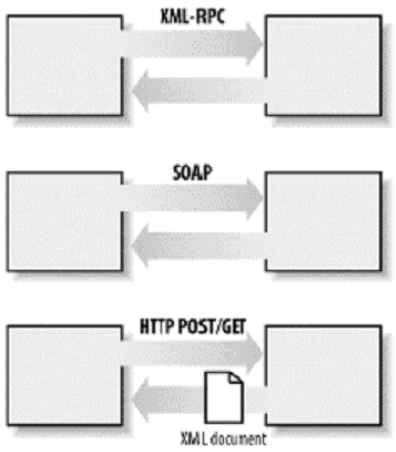

Figura 4.3: Envío de mensajes XML en Servicios Web

El servicio de autenticación/autorización utiliza como protocolo de comunicación una implementación de SOAP, lo cual se explicará más adelante.

Un Servicio Web tiene dos propiedades adicionales:

- se auto-describe: un Servicio Web expone una interfaz pública (gramática XML para identificar todos los métodos públicos, sus argumentos y valores de retorno), e incluye documentación para que otros desarrolladores puedan integrarlo fácilmente.

- es fácil de descubrir: existen mecanismos relativamente sencillos para publicarlo, y por medio de los cuales las partes interesadas pueden encontrar el servicio y pueden localizar su interfaz pública.

En definitiva, el servicio de autenticación/autorización **IAM**, por tratarse de un Servicio Web, tiene las siguientes características:

- Está disponible sobre Internet o Intranet (red privada).
- Usa un envío estándar de mensajes XML.
- No está ligado a ningún Sistema Operativo ni lenguaje de programación.
- Se auto-describe mediante una gramática XML común.
- Es descubierto mediante un mecanismo simple de búsqueda.

El servicio provee ciertas funciones que implementan el mecanismo de autenticación/autorización. La interfaz pública es la siguiente:

- ***enabledPassport***: permite determinar si el servicio soporta manejo de sesión única.
- ***login***: valida que el nombre de usuario y contraseña ingresados sean correctos.
- ***validate***: valida la sesión y verifica que el usuario tenga acceso a la aplicación solicitada.
- ***logout***: finaliza la sesión.

Las funciones principales que implementan el mecanismo son ***login*** y ***validate***. Estas trabajan conjuntamente en un proceso de dos fases, en donde primero se valida la información (autenticación) y se crea la sesión en caso de tener éxito, y como segundo paso se verifica que la sesión sea válida y que el usuario cuente con los privilegios para acceder a la aplicación solicitada (autorización).

Como se mencionó al inicio de este capítulo, el servicio de autenticación/autorización se comunica con un módulo externo, el cual determina si la información es correcta o no.

Esta comunicación se establece a través de canales específicos y bajo determinadas interfaces que debe respetar e implementar el módulo externo, las cuales se ven en la Fig. 4.4

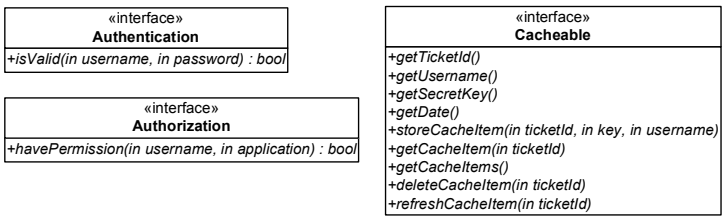

Figura 4.4: API de integración Servicio **IAM** - Módulo Externo de Autenticación

La interfaz **Authentication** determina si un nombre de usuario y contraseña son válidos; la interfaz **Authorization**, determina si un usuario válido tiene los permisos para acceder a la aplicación solicitada; y la interfaz **Cacheable**, establece el manejo para la información de sesiones, permitiendo realizar los chequeos necesarios, como son la validez y tiempos de expiración.

Esto permite tener varias implementaciones, con fuentes de información diversas, lo que hace que el modelo se ajuste a los distintos escenarios con un costo relativamente bajo. El servicio **IAM** lo único que determina es el flujo de control y establece las reglas de negocio, actuando como un framework, e invocando a los componentes externos cuando así lo requiere.

Es así que, por ejemplo, se podría tener un sistema que realice la validación del usuario, otro que valide los permisos y un tercero que almacene la información de las sesiones.

Esta integración del módulo externo con el modelo **IAM** se verá con mayor detalle en la *Sección 4.6*.

A continuación se explica detalladamente el rol y funcionamiento de cada una de las operaciones provistas por el servicio de autenticación/autorización **IAM**.

El método ***enabledPassport*** determina si el modelo soporta el manejo de sesión única leyendo su configuración, la cual se obtiene cuando se inicia el servicio. **IAM** maneja dos tipos de sesión:

- **Sesión de usuario**: cuando está habilitado, el servicio crea una sesión única para el usuario que accede a una aplicación integrada por primera vez, manteniéndola en su cache, evitando de este modo que el usuario tenga que ingresar su nombre de usuario y contraseña cada vez que quiera acceder a una nueva aplicación. (*Sección 4.2.1*)

- **Sesión de aplicación**: cuando no está habilitado, el servicio crea una sesión cada vez que el usuario accede a una aplicación. (*Sección 4.2.2*)

Cuando el servicio **IAM** recibe un pedido de ***enabledPassport***, interactúa con **IAMMasterConfigFile**, que es el módulo que mantiene la configuración global del modelo. Como resultado se obtiene un true/false en función de esta configuración. Este proceso se ve en la Fig. 4.5.

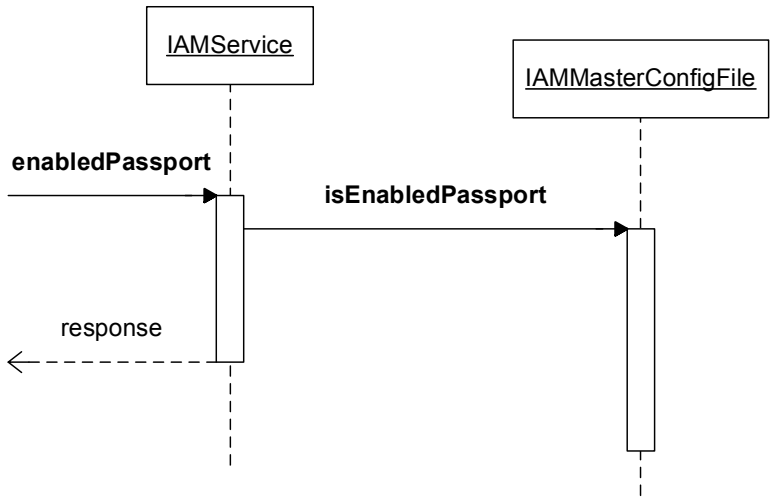

Figura 4.5: Diagrama de interacción - ***enabledPassport***

El método ***login*** realiza el primer paso del mecanismo de autenticación/autorización, es decir la autenticación del usuario, verificando si dicho usuario es válido. Esta función recibe el nombre de usuario y la contraseña, y se la envía al módulo externo de autenticación. Si dicha información es incorrecta, ya sea porque el usuario no existe o su contraseña no coincide, se retorna un código indicando este caso, y el usuario no tiene acceso a la aplicación; en caso de que la autenticación resulte exitosa, se genera la información necesaria para establecer una sesión, independientemente del tipo de sesión que sea. Esta información consta de un identificador unívoco de sesión ***"ticketId"*** y una clave secreta de encriptación ***"secretKey"***. En conjunto con el nombre de usuario, es enviada al módulo externo encargado de mantener la información relativa a las sesiones del modelo. Esta interacción y los objetos involucrados se ven en la Fig. 4.6.

La respuesta obtenida consta del identificador unívoco de sesión, el nombre de usuario, la contraseña encriptada con la clave secreta generada para la sesión y un código indicando si el proceso resultó exitoso o no.

El método ***validate*** realiza el segundo paso del proceso, validando la información de sesión y verificando si el usuario tiene permiso para acceder a la

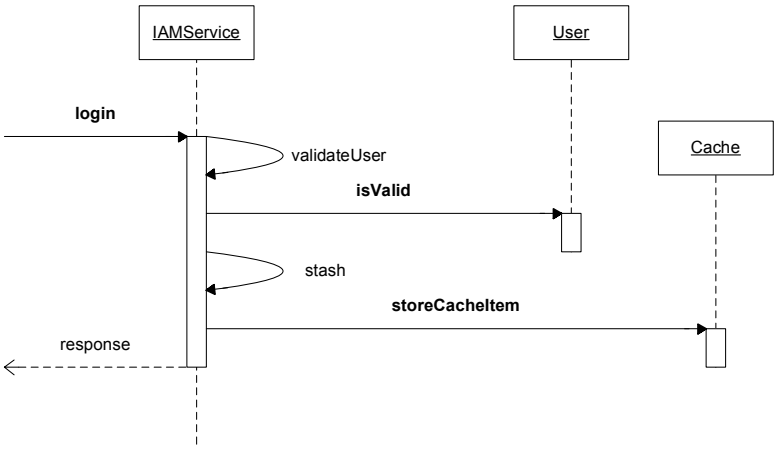

Figura 4.6: Diagrama de interacción - ***login***

aplicación solicitada. Esta función recibe la información de sesión, junto con la aplicación. Primero se intenta recuperar la sesión para verificar si todavía está activa. Si esta información no está disponible, significa, o bien que la sesión expiró o que la sesión recibida está corrupta, en cuyo caso se devuelve el código de error correspondiente. Si la sesión es válida, se desencripta la contraseña recibida con la clave secreta de sesión almacenada en el cache del modelo, y conjuntamente con el nombre de usuario, se vuelve a chequear para asegurar que la información no se corrompió entre los dos pasos. En este momento se tiene certeza de que la sesión es válida, por lo que se procede a verificar los permisos del usuario, para determinar si tiene acceso a la aplicación solicitada. En la Fig. 4.7 se ve la interacción entre los objetos involucrados en esta operación.

Si este proceso resulta exitoso, se actualizará la información de sesión "cacheada", de forma tal de mantenerla activa. Esta función puede retornar varios códigos, según el paso de la validación que falló. Los posibles resultados pueden ser: la sesión ha expirado, nombre de usuario o contraseña incorrectos, permisos insuficientes o validación exitosa. En este último caso, el usuario podrá acceder a la aplicación solicitada.

El método ***logout*** elimina la información de sesión "cacheada" por el modelo. Esta función recibe el identificador de sesión ***"ticketId"***, e invalida dicha sesión, indicando de este modo que el usuario ha finalizado su tarea. Este proceso se muestra en la Fig. 4.8.

Esta función indica que la sesión ha finalizado, tanto sea en el caso de una sesión de aplicación o una sesión de usuario. Si la sesión ya ha expirado, no tiene efecto alguno.

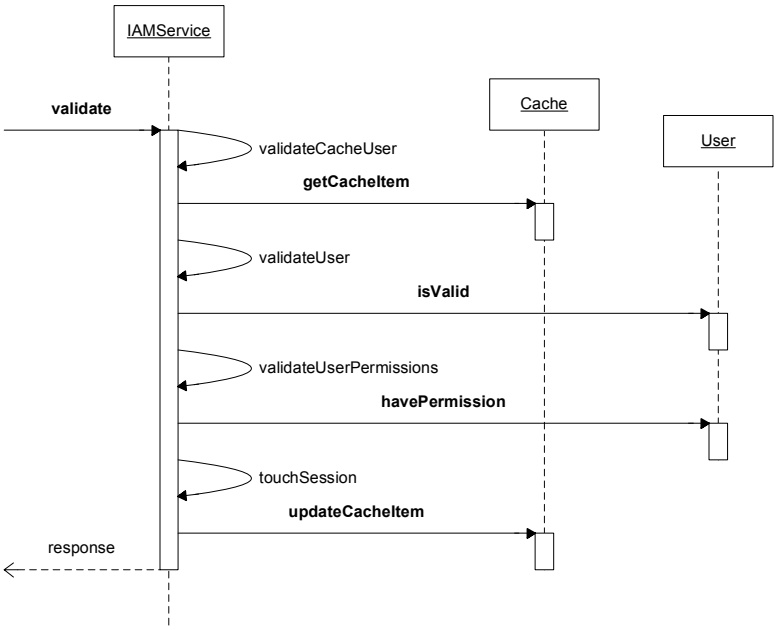

Figura 4.7: Diagrama de interacción - ***validate***

Mediante estas cuatro funciones públicas, las librerías de integración se comunican con el servicio, permitiendo la integración de las diferentes aplicaciones de Groupware con el modelo **IAM**.

En la Fig. 4.9, se muestran los diferentes componentes que forman parte del servicio de autenticación/autorización.

Como se puede apreciar en el diagrama de clases de la Fig. 4.9, existen una serie de componentes que también participan activamente del proceso.

En el paquete ***"ar.edu.iam.services.transport"*** se encuentran los componentes que proveen la estructura para el transporte de la información entre el servicio de autenticación/autorización **IAM** y las librerías de integración (*Sección 4.4*). En el paquete ***"ar.edu.iam.model"*** se definen los componentes del módulo externo de autenticación, los cuales deben implementar la funcionalidad descripta en el paquete ***"ar.edu.iam.interfaces"***, como se explica en la *Sección 4.6*. Por último en el paquete ***"ar.edu.iam.system"*** se encuentran los componentes que inicializan el servicio **IAM**, mantienen su configuración, y administran la información "cacheada" por el modelo.

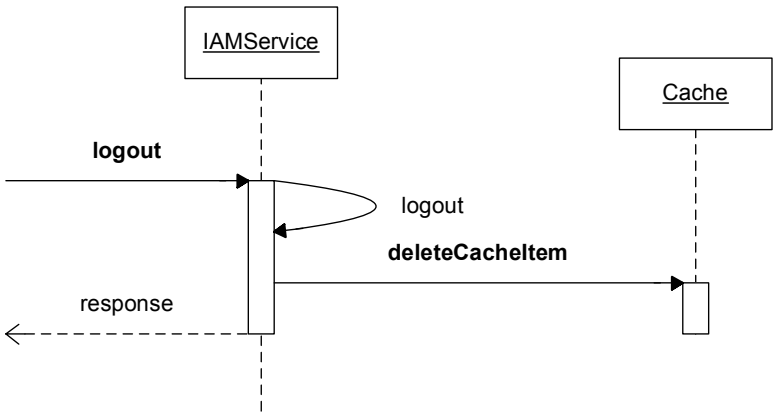

Figura 4.8: Diagrama de interacción - ***logout***

4.2 Módulo para el manejo de sesión única

El módulo para el manejo de sesión única permite a los usuarios establecer, la primera vez que acceden a una aplicación integrada, una sesión única de usuario, de forma tal que no se requiera ingresar nuevamente el nombre de usuario y la contraseña para utilizar las demás aplicaciones.

Como se mencionó anteriormente, el modelo es configurable, permitiendo manejar, tanto sesión única de usuario como sesión de aplicación. Si se habilita esta característica, y el usuario acepta determinadas condiciones de seguridad requeridas por el módulo, se podrá establecer una única sesión de usuario, de forma tal que las aplicaciones accedidas no requieran ingresar nuevamente el nombre de usuario y la contraseña; si esta característica se encuentra deshabilitada o bien no se aceptan las condiciones de seguridad antes mencionadas, cada vez que un usuario intente acceder a una aplicación integrada con el modelo **IAM**, esta le pedirá que ingrese su nombre de usuario y contraseña nuevamente, y se creará en este caso una sesión de aplicación.

En las secciones siguientes se explicarán con mayor detalle los dos tipos de sesión que se pueden establecer con el modelo.

4.2.1 Sesión de aplicación

Cuando un usuario intenta acceder a una determinada aplicación que está integrada al modelo, debe ingresar su nombre de usuario y contraseña. Esta información es interceptada por un módulo cliente de **IAM**, el cual intenta establecer comunicación con el servicio de autenticación/autorización. Una vez establecida la comunicación, dicho módulo envía una petición de ***login***;

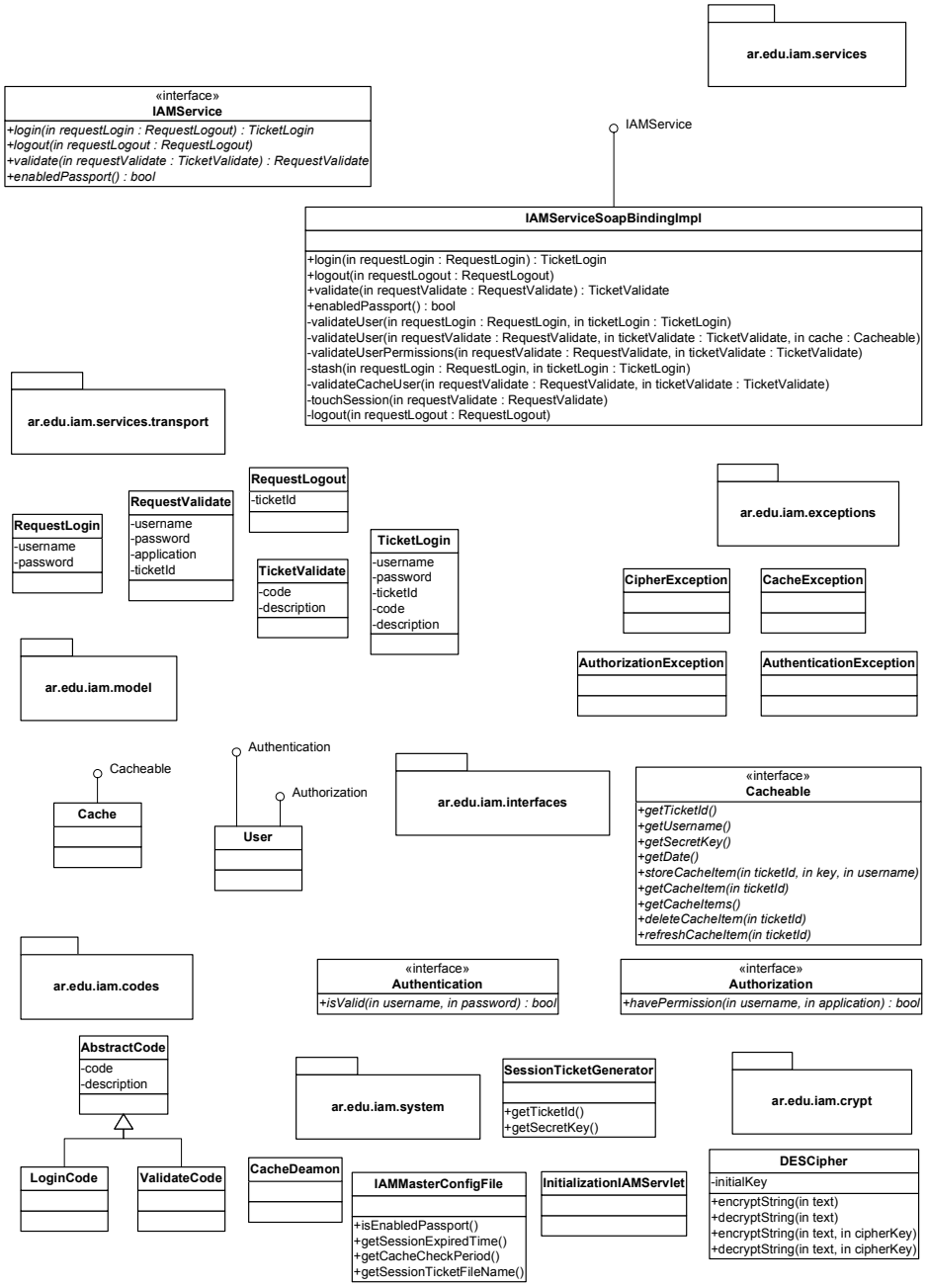

Figura 4.9: Diagrama de Clases - Servicio de Autenticación/Autorización **IAM**

si la validación falla, el usuario no puede ingresar a la aplicación; en caso de tener éxito, se establece la sesión de aplicación con el modelo, y se procede a invocar a ***validate***, enviando la información de sesión obtenida junto con la aplicación. En caso de que la sesión sea válida, la información sea correcta y el usuario tenga los permisos suficientes, podrá acceder a la aplicación; en caso contrario se le informa al usuario cuál fue el error.

4.2.2 Sesión de usuario

Cuando un usuario intenta acceder a una determinada aplicación que está integrada al modelo, un módulo cliente interceptará esta acción, e intentará establecer comunicación con el servicio de autenticación/autorización **IAM**. Como primer paso se envía un mensaje de ***enabledPassport*** al servicio, para saber si el modelo soporta esta característica; en caso afirmativo, dicho módulo intenta recuperar la información de sesión almacenada en la máquina del cliente.

Cuando se quiere utilizar esta función de *"passport"*, el usuario deberá acceder a otro módulo, provisto por el modelo, en donde ingresará su nombre de usuario y contraseña. Este módulo es **IAM Login**.

IAM Login es una aplicación JWS (Java Web Start), independiente del Sistema Operativo utilizado, que tiene las siguientes características: se instala automáticamente en la máquina del usuario; cada vez que se accede, chequea que la versión esté actualizada con respecto a la que está en el servidor; instala, en caso de que no se encuentre ya instalado, el soporte para que pueda funcionar.

Cuando se quiere utilizar esta aplicación, la misma requerirá que el usuario acepte determinadas condiciones de seguridad para poder tener acceso a ciertos recursos locales, necesarios para el correcto funcionamiento del mecanismo de autenticación/autorización.

La aplicación **IAM Login** se muestra en la Fig. 4.10.

A través de esta aplicación, el usuario ingresa su nombre de usuario y contraseña, y se invoca a la función ***login*** del servicio **IAM**, la cual procederá como se explicó anteriormente. Si esta validación tiene éxito, se guarda la información de sesión directamente en la máquina del usuario. A partir de este momento, cuando el usuario quiera acceder a cualquiera de las aplicaciones integradas con el modelo **IAM**, el módulo cliente de cada aplicación, reconocerá esta información, y no se le pedirá que ingrese nuevamente su nombre de usuario y contraseña. Este módulo cliente invocará a ***validate*** para verificar dicha información y los permisos del usuario.

La aplicación **IAM Login**, es la encargada además, de mantener la sesión

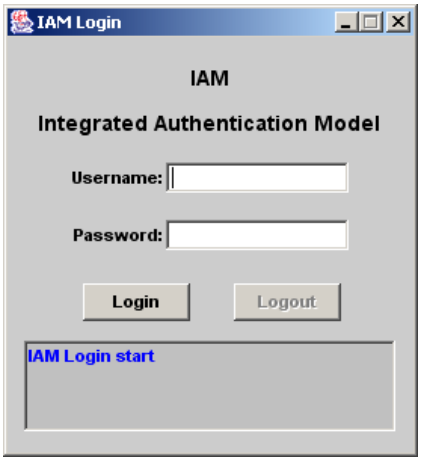

Figura 4.10: GUI del Módulo de manejo de sesión única - **IAM Login**

del usuario activa, y verificar que esta no esté corrupta, haciendo llamados a ***validate*** cada intervalos regulares de tiempo. De esta forma, si la sesión expira o la información de sesión en la máquina del usuario es alterada de alguna manera, la aplicación se entera e invalida dicha sesión. Del mismo modo, al hacer ***logout***, se elimina la sesión, y se borra la información de la máquina del usuario.

En conclusión, si el soporte para *"passport"*no está habilitado, el usuario seguirá utilizando las aplicaciones normalmente, con la ventaja de tener un único nombre de usuario y contraseña para acceder a todas ellas. Si esta característica esta habilitada, se le está mostrando al usuario que se está utilizando algún mecanismo diferente para hacer la autenticación.

4.3 Herramienta de Administración

La herramienta de administración **IAM Admin**, proporciona al administrador de aplicaciones, una forma centralizada, ágil y sencilla de manejar toda la información relativa al mecanismo de autenticación/autorización de **IAM**. Es una aplicación web, en donde se pueden administrar las distintas aplicaciones integradas al modelo **IAM**, los usuarios y los permisos sobre las mismas.

Como se puede apreciar en la Fig. 4.11, desde esta página también se tiene acceso al módulo de manejo de sesión única **IAM Login**.

La herramienta de administración contempla dos tipos de usuarios: ***Usuario Administrador*** y ***Usuario de Aplicación***.

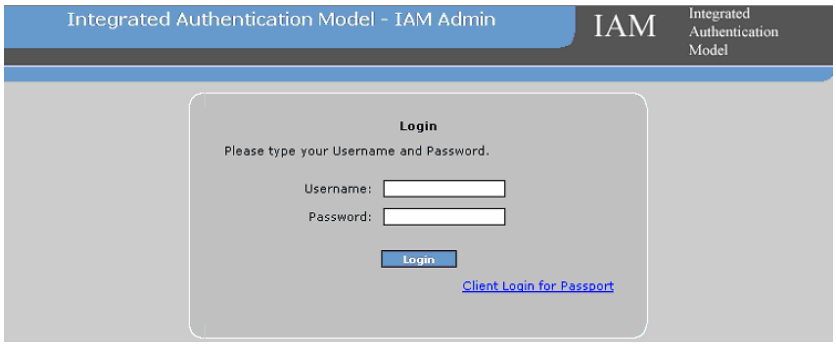

Figura 4.11: Página de acceso a **IAM Admin**

El ***Usuario Administrador*** tiene habilitada opciones para la administración de usuarios (Fig. 4.12) y aplicaciones (Fig. 4.13).

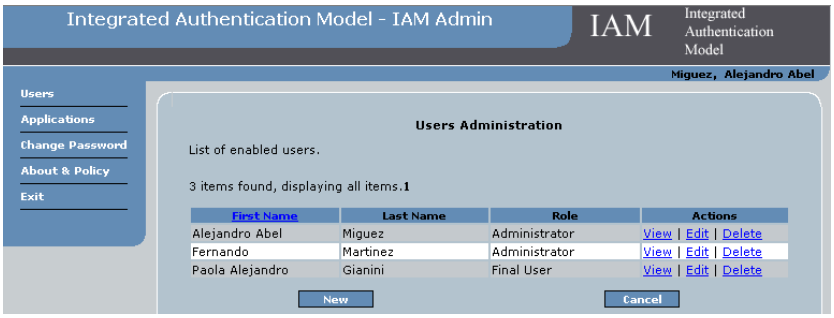

Figura 4.12: Listado de Usuarios - **IAM Admin**

En las dos figuras (Fig. 4.12 y Fig. 4.13) se muestran las aplicaciones integradas al modelo **IAM** y los usuarios habilitados para autenticarse utilizando el modelo **IAM**. Como se ve, es posible agregar, eliminar y modificar los usuarios y las aplicaciones.

En las figuras (Fig. 4.14 y Fig. 4.15) se pueden apreciar las distintas propiedades de los usuarios y aplicaciones que se pueden modificar.

La administración de los permisos de los usuarios sobre las aplicaciones se puede ver en la Fig. 4.14. Aquí también se cuenta con la posibilidad de restablecer la contraseña del usuario, según las políticas de la herramienta.

El ***Usuario de Aplicación***, podrá modificar sus opciones personales (nombre, apellido y e-mail), cambiar su contraseña y visualizar su perfil. En el mismo, aparecerá un listado de las aplicaciones para las cuales tiene permiso. En las figuras (Fig. 4.16 y Fig. 4.17) se muestra esta funcionalidad.

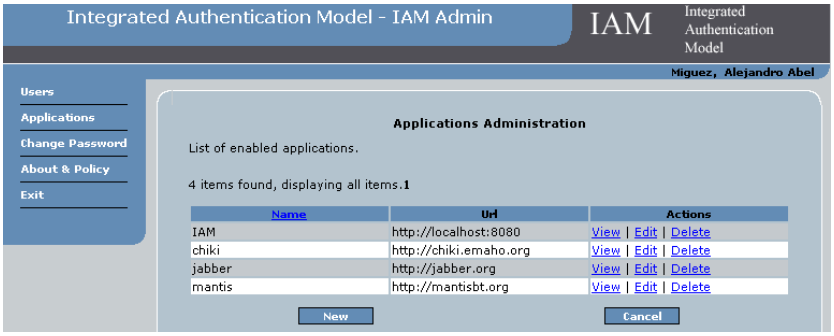

Figura 4.13: Listado de Aplicaciones - **IAM Admin**

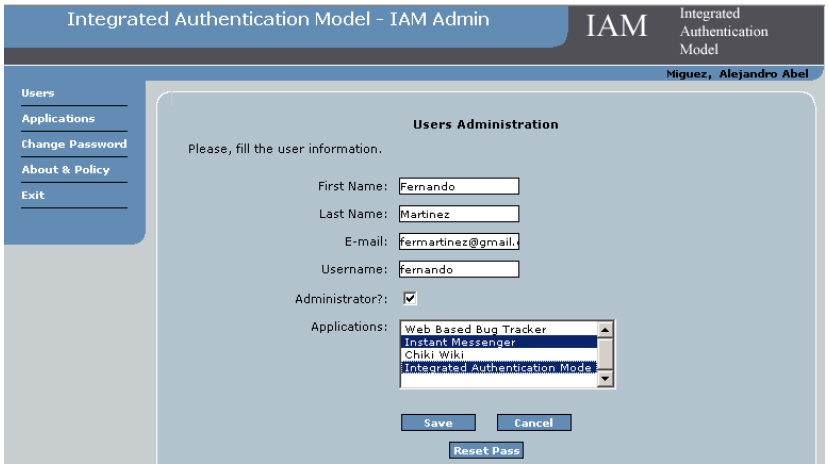

Figura 4.14: Modificación de Usuarios - **IAM Admin**

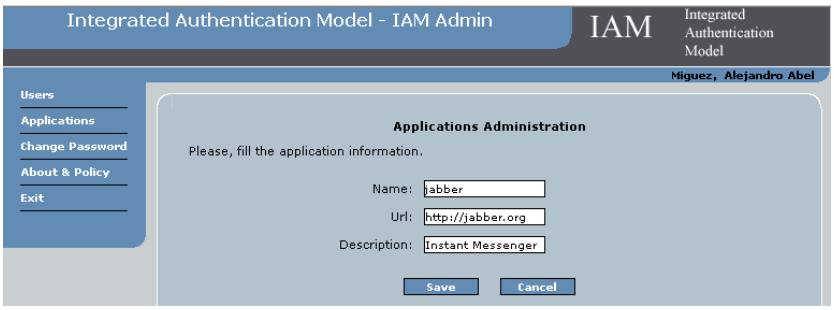

Figura 4.15: Modificación de Aplicaciones - **IAM Admin**

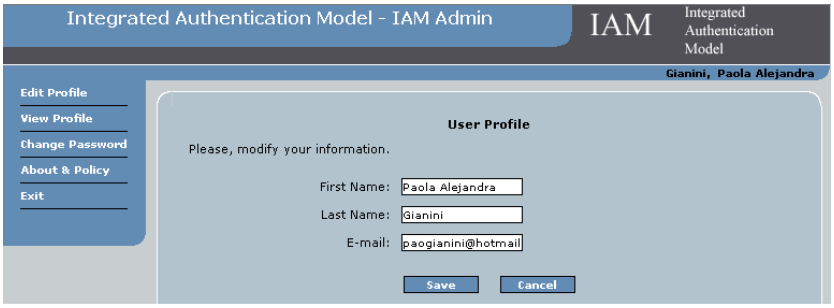

Figura 4.16: Modificación del perfil - **IAM Admin**

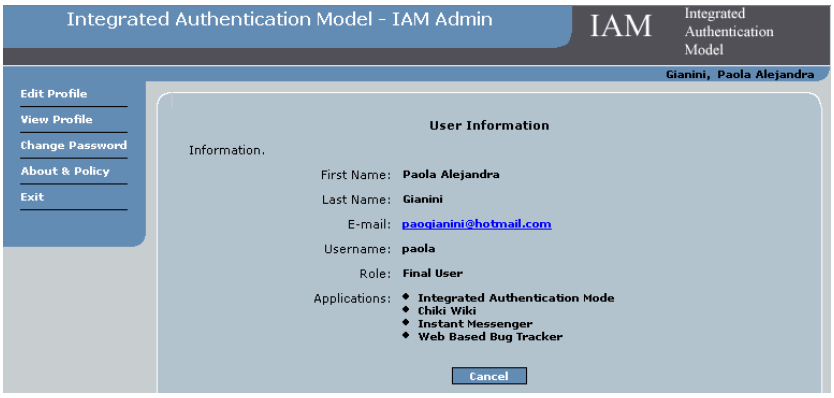

Figura 4.17: Visualización del perfil - **IAM Admin**

Es importante destacar, que la aplicación de administración interactúa con el módulo externo a través de interfaces bien definidas. Estas interfaces se muestran en la Fig. 4.18.

La interfaz **UpdatableUser** define los atributos básicos y las operaciones requeridas para administrar los usuarios del modelo. La interfaz **UpdatableApplication** define atributos y operaciones similares para las aplicaciones.

4.4 Librerías de integración

Las librerías permiten la fácil integración de las aplicaciones de Groupware, estableciendo la comunicación con el servicio de autenticación/autorización **IAM**. El modelo provee estas librerías para posibilitar la integración de aplicaciones desarrolladas en los lenguajes más populares, como son Java, PHP, C/C++. Estas librerías pueden ser utilizadas para integrar, tanto aplicaciones Web, como aplicaciones Cliente/Servidor.

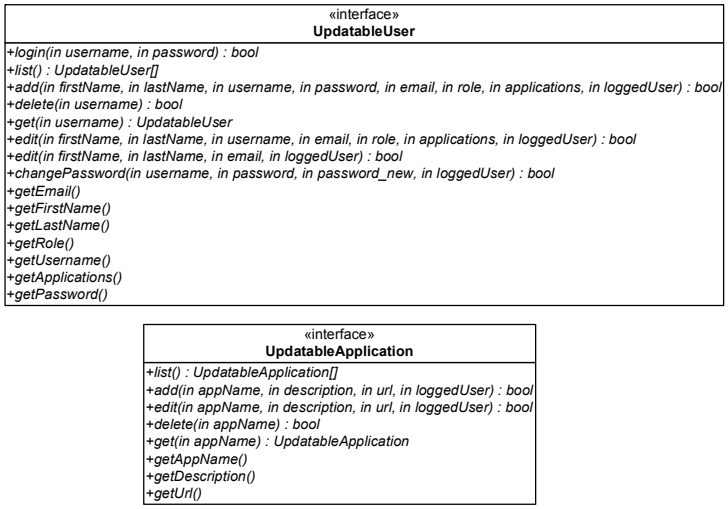

Figura 4.18: API de integración **IAM Admin** - Módulo Externo

La API de las librerías consta básicamente de cuatro funciones:

- ***iam_can_use_passport***: determina si el servicio **IAM** está configurado con soporte para el manejo de sesión única *"passport"* y si existe una sesión de usuario iniciada por parte del cliente.
- ***iam_get_passport_info***: recupera la información de *"passport"* almacenada en la máquina del cliente.
- ***iam_ensemble_password***: confecciona la contraseña de manera de saber si se está utilizando *"passport"* o no para la autenticación, y hacer que la función ***iam_authenticate*** sea genérica con respecto al tipo de autenticación.
- ***iam_authenticate***: realiza la autenticación y autorización del usuario.

Las función principal de la API es ***iam_authenticate*** y es la encargada de comunicarse con el servicio **IAM** para que el mismo lleve a cabo el proceso de autenticación/autorización y así determinar si el usuario tiene permiso para utilizar la aplicación que está solicitando.

A continuación, se verá más en detalle el rol y funcionamiento de cada una de las funciones provistas por la API de las librerías de integración:

iam_can_use_passport: esta función se comunica con el servicio **IAM**, a través de la función ***enabledPassport***. En este punto, el cliente conoce si el servicio está configurado con soporte para *"passport"*, y de ser así, entonces

debe verificar que el usuario haya iniciado una sesión de usuario con el modelo **IAM**. Para ver que esto sea así, se intenta determinar la existencia de la información de *"passport"* que está almacenada en el directorio del usuario, la cual es generada por la aplicación **IAM Login**. Si la misma existe, entonces quiere decir que el usuario ya inició una sesión con el modelo **IAM**, entonces se determina que el uso de *"passport"* para la autenticación/autorización está permitido. En el caso de que el servicio no esté configurado con soporte para *"passport"*, entonces se procede a indicarle al cliente que el uso de *"passport"* no está permitido.

iam_get_passport_info: una vez que se determina que el uso de *"passport"* está permitido, entonces se utiliza esta función, la cual se encarga de obtener la información de *"passport"* almacenada en el directorio del usuario. Cabe destacar que se debe utilizar esta función solamente si es seguro que la sesión de usuario fue establecida con el modelo **IAM**, en caso contrario, se estará intentando obtener información que todavía no se generó en la máquina del cliente.

iam_ensemble_password: luego de obtenida la información de la sesión de usuario, se procederá a invocar a esta función, cuya responsabilidad es la de construir la contraseña que será la utilizada para enviar al servicio **IAM**. La contraseña generada por esta función tendrá una estructura particular. Se vió en secciones anteriores, que el uso de *"passport"* determina que la información enviada al servicio **IAM** a la hora de llevar a cabo el proceso de autenticación/autorización no es la misma, entonces, lo que hace esta función de la API, es generar la información con el formato adecuado para que la función ***iam_authenticate***, pueda determinar qué tipo de sesión es la que se está estableciendo, así como también contar con toda la información necesaria para enviar al servicio **IAM**.

iam_authenticate: esta función, es la encargada de comunicarse con el servicio **IAM** para determinar si el usuario efectivamente tiene permiso para utilizar la aplicación que está solicitando. El comportamiento de la misma, es diferente en caso de que se esté utilizando *"passport"* o no, entonces debe contar con alguna manera de determinar si se está intentando autenticar/autorizar una sesión de aplicación o una sesión de usuario. Si se está utilizando *"passport"*, entonces la contraseña enviada como parámetro a esta función, tendrá una estructura particular, si no, será solamente una cadena de caracteres sin ninguna estructura. Dicho esto, la función ***iam_authenticate*** determinará el tipo de sesión dependiendo de la estructura subyacente de la contraseña que se le haya enviado.

Si la sesión es de aplicación, entonces la función llevará a cabo el proceso en dos fases que se vió en secciones anteriores: primero llamará al servicio **IAM** a través de la función ***login*** para realizar la autenticación del usuario. Si el usuario no es válido, entonces se procederá a informar al cliente de esta situación. En caso de que el usuario sea válido, la función ***login*** retornará información acerca de la sesión de aplicación recientemente establecida con el modelo **IAM** y ésta será utilizada para llamar a la función ***validate*** del servicio **IAM**. Luego, la función ***iam_authenticate*** retornará éxito o fracaso dependiendo si el usuario tiene permisos suficientes para utilizar la aplicación solicitada.

4.5 Comunicación

La comunicación entre los módulos clientes (librerías) y el servicio de autenticación/autorización **IAM**, se realiza sobre un canal seguro (**HTTPS**). De este modo, la información sensible que viaja hacia el servicio (como son el nombre de usuario y contraseña de un determinado usuario dentro de la organización), en caso de que sea interferida por algún intruso, no podrá ser obtenida de una forma fácil.

La seguridad del canal, se consigue con el uso de certificados, haciendo que los clientes se autentiquen contra el servicio (para que solo los clientes habilitados puedan comunicarse), y a su vez el servicio se autentica contra los clientes (así los clientes verifican que el servicio que están invocando sea el correcto).

Los certificados normalmente contienen información como su nombre, empresa, departamento, dirección de correo, ciudad, país, y otros; lo que permite sofisticados esquemas de control de acceso basándose en uno de estos atributos o en una combinación de varios.

La información que viaja por la red es cifrada mediante una clave secreta. Cuando se emplea la misma clave en las operaciones de cifrado y descifrado, se dice que el criptosistema es simétrico o de clave secreta. Cuando se utiliza una pareja de claves para separar los procesos de cifrado y descifrado, se dice que el criptosistema es asimétrico o de clave pública.

Una clave, la privada, se mantiene secreta, mientras que la segunda clave, la pública, puede ser conocida por todos. De forma general, las claves públicas se utilizan para cifrar y las privadas, para descifrar. El sistema tiene la propiedad de que a partir del conocimiento de la clave pública no es posible determinar la clave privada. Los criptosistemas de clave pública, aunque más lentos que los simétricos, resultan adecuados para las funciones de autenticación, distribución de claves y firmas digitales.

Para que la autenticación con certificados funcione, las pedidos (requests) deben ser precedidas por "https://".

En definitiva, los certificados cumplen dos funciones importantes: la de cifrar los datos y la de autenticar entidades. De esta forma, se asegura la confidencialidad e integridad de la información que viaja por la red.

Sin embargo, este uso de certificados tiene la falencia de que se pueden obtener (robar) y permitir que un determinado cliente (provisto por una tercera parte), se comunique con el servicio **IAM**, teniendo de esta forma acceso a información sensible. Para solucionar este problema, los clientes además presentan al servicio un nombre de usuario y contraseña, conocido como **Autenticación Básica**, permitiendo de esta forma que la comunicación sólo se pueda realizar satisfactoriamente teniendo el certificado correcto y el nombre de usuario y contraseña adecuados.

La **Autenticación Básica** funciona de la siguiente forma:

Cuando el usuario accede a un recurso del servidor Web protegido mediante **Autenticación Básica**, tiene lugar el siguiente proceso:

1. el usuario debe informar su nombre y contraseña.
2. se intenta establecer una conexión con el servidor utilizando esta información.
3. si el servidor rechaza la información de autenticación, la conexión no tiene lugar.
4. cuando el servidor Web verifica con éxito los datos de autenticación, se establece la conexión de acceso al recurso protegido.

En conclusión, los componentes del modelo **IAM** se comunican utilizando **HTTPS + Autenticación Básica**, estableciendo así un nivel de seguridad suficiente para una Intranet.

4.6 Módulo Externo de Autenticación

El módulo externo de autenticación, como se ve en la Fig. 4.1, es el que provee toda la información necesaria para el completo funcionamiento del mecanismo de autenticación/autorización. Este módulo se adapta al modelo **IAM** según las interfaces (APIs de integración) vistas en las figuras Fig. 4.4 y Fig. 4.18, permitiendo la comunicación con el servicio de autenticación/autorización **IAM** y la herramienta de administración **IAM Admin**.

El módulo externo debe contar con uno o varios repositorios, donde almacena la información utilizada por el modelo **IAM** (usuarios, aplicaciones,

permisos, sesiones establecidas con el modelo) y tener un protocolo mediante el cual comunicarse con dicho/s repositorio/s. La flexibilidad de este módulo reside en la posibilidad de utilizar diferentes medios de persistencia, como por ejemplo: Bases de Datos, LDAP o archivos XML.

Para que el modelo **IAM** funcione correctamente, éste módulo debe garantizar la confiabilidad e integridad de la información, dado que es el encargado de manejar datos sensibles del modelo. Es importante tener en cuenta al momento de elegir el medio de persistencia, que el mismo cuente con mecanismos de seguridad suficientes para proteger dicha información contra accesos no autorizados.

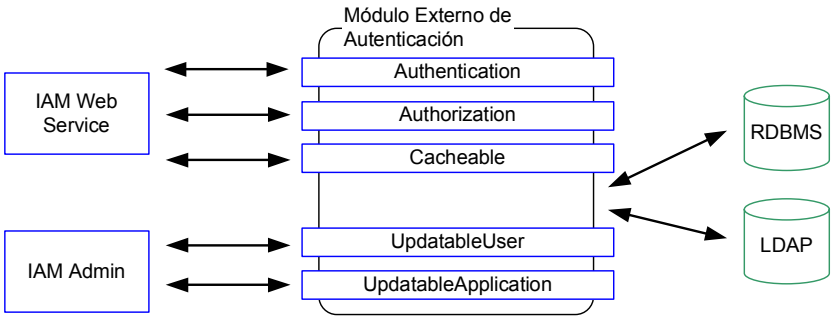

Figura 4.19: Interacción modelo **IAM**- Módulo Externo

Como se puede apreciar en la figura Fig. 4.19, el servicio de autenticación/autorización **IAM**, se comunica con el módulo externo a través de las interfaces **Authentication**, **Authorization**, y **Cacheable**; mientras que la comunicación de dicho módulo con **IAM Admin** se logra a través de las interfaces **UpdatableUser** y **UpdatableApplication**.

Capítulo 5

Implementación

En este capítulo se describen las decisiones de diseño tomadas durante la etapa de análisis del problema, se muestran las tecnologías utilizadas y se explica la implementación del modelo **IAM**.

5.1 Decisiones de diseño

5.1.1 Arquitectura

El problema a resolver, ya visto en el *Capítulo 2*, implicaba abstraerse de las aplicaciones a integrar y de las arquitecturas y plataformas que utilizan. Entonces se debía proveer una forma sencilla de integrar estas aplicaciones con un único mecanismo de autenticación.

En este momento se opta por desarrollar el servicio de autenticación/autorización como un Servicio Web, que debido a sus características lo convierten en la tecnología más adecuada para este modelo. Alrededor de este servicio, se construyó el resto del modelo **IAM**.

Otro punto importante, se presentó al definir las responsabilidades del servicio de autenticación/autorización **IAM**. Es este caso se determinó, que el servicio sólo debía establecer el flujo y las reglas de negocio del mecanismo de autenticación, delegando la responsabilidad de cómo hacer las validaciones correspondientes al módulo externo, el cual tiene acceso a la información del repositorio. Es aquí donde se definieron determinadas interfaces (API), que el módulo externo debe respetar para poder integrarse al modelo **IAM**.

Otra decisión a tomar, fue cómo permitir el manejo de sesión única y de qué forma. En este momento se decide delegar a un módulo particular (**IAM Login**), que se ejecutará en la máquina del cliente y administrará la información necesaria para el servicio **IAM** y las aplicaciones integradas.

Con respecto a la comunicación, por tratarse de información sensible, se debía manejar con cautela. Es por eso que se decide que toda la comunicación del modelo se haga a través de un canal cifrado, dificultando de este modo recuperar la información transmitida por la red ante cualquier intrusión.

Todas estas decisiones, junto con la posibilidad de brindar librerías para integrar rápidamente las aplicaciones, y proveer una forma de administrar la información relativa a los usuarios, determinaron la arquitectura vista en la *Sección 4.1*.

5.1.2 Mecanismo de Autenticación

Al momento de definir el mecanismo de autenticación, por tratarse del punto crítico del sistema, se tuvieron en cuenta varios aspectos: determinar si un usuario es válido para el modelo, verificando su identidad; determinar si posee los privilegios suficientes para acceder a la aplicación solicitada; definir la información necesaria para establecer la sesión; controlar los distintos errores que puede generar el módulo externo, y definir las políticas que determinan qué acciones tomar ante estos errores e informarle al usuario de los problemas ocurridos.

Al definir las reglas del mecanismo de autenticación, se diferenciaron claramente dos fases: la primera, Autenticación, es decir, determinar si un usuario, a través de su nombre de usuario y contraseña, es válido para el modelo. Si la Autenticación resulta exitosa, se establece la sesión, y se envía esta información al cliente; la segunda, Autorización, se valida nuevamente la información del usuario y de sesión, y se verifica que el usuario tenga los permisos para acceder a la aplicación solicitada.

Para definir el manejo de errores se crearon una serie de Excepciones y Códigos de Error para las dos fases, permitiéndole de este modo a las librerías de integración, informar de un modo amigable el problema al usuario.

5.1.3 Conceptos de Sesión y Ticket

Sesión y **Ticket** son dos conceptos abstractos del modelo, fundamentales para entender su comportamiento. En el momento que un usuario se ha logueado satisfactoriamente al modelo, queda establecida una sesión. El usuario se puede loguear, o bien directamente desde una aplicación integrada, o utilizando el Módulo de manejo de sesión única **IAM Login** visto en la *Sección 4.4*. Es por esto que se definieron dos tipos de sesión:

Sesión de aplicación: es aquella que se establece entre el modelo y el usuario para una determinada aplicación.

Sesión de usuario: es compartida por todas las aplicaciones integradas.

En este momento se define qué información caracterizará a una sesión, independientemente del tipo que sea. Es aquí donde se introduce al modelo el concepto de **Ticket**, que es la información generada en la fase de Autenticación y validada posteriormente en la fase de Autorización. Este Ticket esta compuesto de: identificador univoco de sesión (denominado *ticketId*), el nombre del usuario, y la contraseña encriptada con una clave secreta de sesión (denominada *secretKey*).

Cuando se establece una sesión de usuario, es decir, se está utilizando el mecanismo de sesión única, la información del Ticket es almacenada en la máquina del cliente, y es la que posteriormente será validada en los futuros accesos a las demás aplicaciones integradas. Esta información es administrada por el Módulo de manejo de sesión única.

La información del Ticket (ticketId, secretKey) es generada por el modelo **IAM**, y su administración y validación la realiza el módulo externo.

5.1.4 Seguridad de las Contraseñas

Las contraseñas representan un punto posible de vulnerabilidad para el modelo **IAM**, dado que esta información se expone al módulo externo de autenticación, y es almacenada en la máquina del cliente. Es aquí donde se decidió que las contraseñas enviadas desde el modelo estén cifradas. Para esto se definieron dos políticas de cifrado dependiendo de a dónde se envía la información. Cuando la contraseña es pasada al módulo externo de autenticación, es encriptada con una clave única de **IAM**; cuando es la contraseña del Ticket, es cifrada con una clave específica para la sesión establecida.

El mecanismo de cifrado elegido fue DES [9], que es un algoritmo simétrico basado en claves de 64 bits.

La información en la máquina del cliente es almacenada en el directorio por defecto del usuario (el cual depende del Sistema Operativo), de forma tal que sólo dicho usuario o un administrador tiene acceso a esa carpeta. Esto es manejado por la políticas de seguridad establecidas por la organización.

5.2 Tecnologías

En esta sección se presentan las tecnologías utilizadas en el desarrollo del modelo **IAM**.

En cuanto a la programación del Servicio Web **IAM** del modelo se optó por el lenguaje Java [10] por varias razones: provee un soporte multiplata-

forma de las aplicaciones, que son capaces de funcionar indistintamente sobre diversas plataformas (Sun, IBM, Compaq) y Sistemas Operativos (Windows, Linux, Solaris), dado que los programas son interpretados por una máquina virtual JVM (Java Virtual Machine) nativa del Sistema Operativo; es un lenguaje orientado a objetos, y permite desarrollar sistemas Desktop y Web extensibles, modulares y reusables con menor costo.

El protocolo de comunicación usado entre este servicio y las librerías de integración, es SOAP (Simple Access Object Protocol) [11]. SOAP es un protocolo liviano de intercambio de información en un ambiente distribuido. Está basado en XML y consiste de tres partes: un encabezado que define un framework para describir qué es un mensaje y cómo se procesa, un conjunto de reglas de codificación para expresar instancias de tipos de datos definidos por la aplicación, y una convención para representar llamadas a procedimientos remotos (RPC) y respuestas.

Existen varias implementaciones del protocolo SOAP. Para esta tesis se utilizó AXIS [12], que es un engine SOAP (framework) para construcción de procesos; incluye un server, soporte para WSDL (Web Service Description Language), plugins para integrar con el Servlet Container Tomcat, utilitarios para generar clases desde el WSDL, y herramientas de monitoreo TCP/IP. Existen dos implementaciones de AXIS: una en Java y otra en C++. Se optó por la implementación de AXIS en Java, para mantener compatibilidad con el resto del desarrollo.

Toda la comunicación entre el servicio y las librerías de integración se realiza sobre un canal seguro (cifrado) **HTTPS**, utilizando certificados y firmando digitalmente con estos las clases del modelo **IAM**, de forma tal que sólo los clientes que posean los certificados de confianza podrán comunicarse con el servidor. Esto se logra configurando el servidor con un certificado que incluye los clientes que son de confianza. Del mismo modo, cada cliente cuenta con su certificado. Para que pueda establecerse conexión entonces, es necesario que los clientes sean de confianza para el servidor y viceversa, para evitar que los clientes le envíen información sensible a una entidad desconocida.

El modelo **IAM**, determina además, que para poder establecer la comunicación entre los clientes y el servicio de autenticación/autorización **IAM**, los clientes deberán proveer información de usuario y contraseña. Esto último es conocido como **Autenticación Básica**, y se logra configurando al servicio como un recurso protegido, sólo accesible si se proporciona el usuario y contraseña correctos. De este modo, no basta con tener los certificados para poder comunicarse con el servicio, si no también contar con el usuario y con-

traseña apropiados.

Otro punto importante, es el de poder detectar fallas y funcionamiento anómalo, no solo durante la etapa de desarrollo, si no también una vez que el sistema se encuentra en producción, y así poder tomar las decisiones correspondientes. La inserción de sentencias de debugging en el código, es una de las técnicas más utilizadas para detectar estas fallas, ya que provee información precisa acerca del contexto de la ejecución de una aplicación. Para esto se utilizó Log4j [13], que es una API que permite el logueo de información de manera muy flexible, a través de un archivo de configuración, de acuerdo a ciertos niveles y categorías.

Además se desarrollaron librerías en los lenguajes más populares de la actualidad (como Java, PHP, C/C++), que posibilitan la fácil integración de aplicaciones al modelo **IAM**. Es en este momento donde se debe tener en cuenta si la aplicación a integrar responde a una arquitectura Web o Cliente/Servidor. Para las aplicaciones Cliente/Servidor, se cuenta con librerías desarrolladas en Java y C/C++; para las aplicaciones Web, en Java y PHP.

En las aplicaciones Web, era necesario contar con un mecanismo para detectar y leer la información de la sesión establecida por el usuario, como ya se explicó en el *Capítulo 4*. Para esto se provee un Applet [14] desarrollado en Java. Un Applet es un programa que puede ser embebido en cualquier página HTML. Cuando se accede a una página que contiene un Applet, el código es transferido al sistema del cliente y ejecutado por la máquina virtual (JVM) del navegador, lo que asegura que el programa se ejecutará de acuerdo a las restricciones de la JVM.

Del mismo modo, se provee un mecanismo para poder loguearse al modelo **IAM**, y persistir la información de sesión en la máquina del cliente, como se vió en la *Sección 4.2.2*. Se cuenta entonces con una aplicación JWS (Java Web Start) [15], independiente del Sistema Operativo utilizado. JWS tiene las siguientes características: se instala automáticamente en la máquina del usuario; cada vez que se accede, chequea que la versión está actualizada con respecto a la que está en el servidor; instala, en caso de que no se encuentre ya instalado, el soporte para que pueda funcionar. Esta aplicación se encuentra firmada digitalmente y requiere acceder a determinados recursos de la máquina del cliente. Si el cliente acepta el certificado presentado por la aplicación, tendrá acceso irrestricto a la máquina.

Con respecto a las tecnologías utilizadas para el desarrollo de la aplicación de administración **IAM Admin**, tenemos un framework MVC (Model

View Controller) Struts [16]. Struts es una capa de control flexible basado en tecnologías estándar como Java Servlets, JavaBeans, ResourcesBoundles, y XML, que provee su propio componente Controlador, que es integrado con otras tecnologías que proveen la parte del Modelo y la Vista. Para el Modelo, Struts puede interactuar con tecnologías de acceso a datos como JDBC, EJB, o Hibernate. Para la Vista, puede trabajar con JSP (Java Server Pages), incluyendo JSTL y JSF, como así también con templates como Velocity y XSLT. Struts permite así, crear un ambiente de desarrollo extensible para las aplicaciones, utilizando estándares y patrones de diseño efectivos. En la aplicación **IAM Admin** se utilizó Struts, Servlets [17], JDBC [18], JSP [19], JSTL [20] y DisplayTags [21].

Las librerías de integración, dependiendo del lenguaje en las que están desarrolladas, utilizan diferentes librerías para comunicarse con el Servicio Web.

En el caso de la librería en C++, se utiliza gSoap [22], que es un kit de desarrollo que permite acceder y consumir un servicio web de manera transparente. Está enteramente desarrollado en C/C++, y provee una interfaz muy útil para mapear XML a tipos de datos de C/C++.

En el caso de PHP, se utilizó NuSoap [23], que consiste en un grupo de clases PHP, para crear y consumir Servicios Web. Si bien esta librería no implementa todas las especificaciones de SOAP 1.1 y WSDL 1.1, el subconjunto que implementa es suficiente para lo que se necesita dentro de este trabajo.

Por último, para el caso de la librería en Java, se utilizó el conjunto de clases provisto por Apache Axis, sucesor de Apache SOAP, que permite generar un conjunto de clases, las cuales, mediante JAX-RPC, consumen los Servicios Web.

5.3 Implementación del modelo IAM

En esta sección se verán en detalle distintos aspectos de la implementación de cada uno de los componentes del modelo **IAM**, las entidades participantes y sus colaboraciones.

5.3.1 Inicio y configuración del modelo IAM

Cuando el modelo **IAM** se inicia, se ejecuta **InitializationIAMServlet**, el cual indica que se debe leer la configuración e iniciar los demonios correspondientes. La configuración del modelo se encuentra en un archivo "iam.cfg", que tiene la siguiente estructura:

```
<IAM>
```

```
    <enabledPassport>true</enabledPassport>
    <expirationSessionTime>28800</expirationSessionTime>
    <cacheCheckPeriod>25200</cacheCheckPeriod>
    <sessionTicketFileName>/sessionTicket.ses</sessionTicketFileName>
</IAM>
```

A continuación se describe la semántica de cada uno de los valores del archivos de configuración:

`enabledPassport`: Determina si el modelo tiene habilitado el soporte para *"passport"*. Los valores permitidos son true o false.

`expirationSessionTime`: Indica el tiempo de expiración de las sesiones de usuario. El valor está expresado en segundos.

`cacheCheckPeriod`: Indica el intervalo de tiempo esperado entre cada chequeo del cache. El valor está expresado en segundos.
Nota: este valor debe ser siempre menor a `expirationSessionTime`.

`sessionTicketFileName`: Indica el archivo donde se mantendrá el último ticketId generado.
Nota: si no existe, se creará automáticamente. No debe ser modificado ni borrado.

El encargado de mantener esta configuración es **IAMMasterConfigFile**, el cual es una instanciación del patrón de diseño *Singleton* [24]. Como se dijo anteriormente, cuando se inicia el servicio se activa un demonio, **CacheDeamon**, que es el encargado de verificar la validez de las sesiones de los diferentes usuarios, de acuerdo a las propiedades *expirationSessionTime* y *cacheCheckPeriod*.

5.3.2 Servicio de autenticación/autorización IAM

En esta sección se verá el componente principal del modelo **IAM**, el cual implementa las cuatro funciones: enabledPassport, login, validate, y logout.

enabledPassport: determina en función de la configuración del modelo, a través de la propiedad ***enabledPassport*** de **IAMMasterConfigFile**, si el manejo de sesión única está activado.

login: recibe el nombre de usuario y contraseña y, luego de cifrar la contraseña con una clave de encriptación única de **IAM**, se los envía al

módulo externo de autenticación, el cual verifica que sean correctos. En caso de tener éxito la validación, se genera la información de sesión (Ticket), con el ticketId, el nombre de usuario, y la contraseña cifrada con una clave secreta de encriptación (secretKey).

El encargado de generar ticketId y secretKey es **SessionTicketGenerator**. Estos dos valores se obtienen a partir de dos métodos: *getTicketId()* y *getSecretKey()*.

- *getTicketId()* permite generar y almacenar atómicamente el nuevo identificador de sesión ticketId, asegurando de esta forma la unicidad de los mismos. El último ticketId generado es guardado en el archivo determinado por la propiedad *sessionTicketFileName*.
- *getSecretKey()* devuelve una clave random de encriptación de 64 bits.

Este Ticket es almacenado por el módulo externo y es devuelto al cliente, junto con el código de retorno de la operación, que puede ser **0**, en cuyo caso la validación resultó exitosa, o **-1** indicando que el usuario no es válido.

validate: recibe la información de sesión (Ticket), junto con la aplicación a la que el usuario está intentando acceder. Como primer paso, verifica que la información de sesión se corresponda con una sesión válida y que esta no haya expirado. Si la sesión es válida, se vuelve a verificar la información del usuario. La contraseña enviada al módulo externo, se construye de la siguiente manera: la contraseña obtenida del Ticket, es descifrada con la clave secreta de encriptación de esa sesión en particular, y luego es cifrada con la clave de encriptación única de **IAM**. De esta forma se controla que el nombre de usuario y contraseña sean correctos. Luego verifica los privilegios del usuario para la aplicación. Retorna distintos códigos en función de las validaciones realizadas: **0**, la validación resultó exitosa; **-1**, la sesión ha expirado; **-2**, usuario inválido; y **-3**, el usuario no tiene permiso para acceder a la aplicación solicitada.

logout: indica la finalización de la sesión, eliminando la información almacenada.

5.3.3 Módulo para el manejo de sesión única

El manejo de sesión única se realiza a través de un módulo especial llamado **IAM Login**. **IAM Login** es una aplicación JWS (Java Web Start), que se

descarga automáticamente en la máquina del usuario, permitiéndole ingresar su nombre de usuario y contraseña, y poder de esta forma establecer una sesión de usuario única con el modelo **IAM**, y acceder a todas las aplicaciones integradas para las cuales se encuentra habilitado sin tener que ingresar esta información nuevamente. Este módulo puede solamente ser utilizado en el caso de que el modelo **IAM** esté configurado con soporte para *"passport"*.

IAM Login tiene el siguiente comportamiento: cuando un usuario ingresa su nombre de usuario y contraseña, se envía el mensaje ***login*** al servicio de autenticación/autorización **IAM**; en caso de que la información suministrada sea correcta, ya se tiene una sesión de usuario establecida con el modelo, y se procede a escribir la información de sesión (Ticket) en la máquina del usuario. De esta forma un archivo "iam.pas" es generado en el directorio por defecto del usuario, con la información del Ticket. En este momento se inicia un demonio, el cual será el encargado de mantener activa la sesión del usuario, enviando cada intervalo regular de tiempo un mensaje de validate al servicio, validando de esta forma que la información en la máquina del usuario no se corrompa.

La aplicación, una vez que se ha establecido la sesión, se guarda esta información. De esta forma, si el Ticket interno de **IAM Login** difiere del Ticket guardado en la máquina del cliente, o el ***validate*** falla, **IAM Login** envía un mensaje de ***logout*** al servicio **IAM**, invalidando de esta forma la sesión del usuario, y eliminando el archivo "iam.pas". Si la aplicación se cierra, también se finaliza la sesión invocando al ***logout*** y eliminando el archivo.

5.3.4 Librerías de Integración

Como ya se mencionó anteriormente, se desarrollaron librerías para integrar aplicaciones Web y Cliente/Servidor implementadas en los lenguajes más populares como son Java, PHP, y C/C++.

Las librerías de integración con el servicio **IAM** respetan la API vista en la *Sección 4.4*. Todas estas librerías son configurables a través de un archivo de configuración "iam_config", que tiene la siguiente estructura:

```
SERVER_PROTOCOL=https
SERVER_HOST=www.iamservice.com
SERVER_PORT=8443
APP_NAME=jabber
BASIC_AUTH=true
SERVICE_NAME=IAMService
```

A continuación se describe la semántica de cada uno de los valores del

archivo de configuración:

`SERVER_PROTOCOL`: Determina el protocolo utilizado para la comunicación. Valores permitidos: http o https.

`SERVER_HOST`: Indica la dirección IP o Url donde se encuentra el servicio **IAM**.

`SERVER_PORT`: Establece el puerto configurado para la comunicación. Este valor depende de la configuración establecida según el protocolo http o https del servidor Web.

`APP_NAME`: Este valor determina el nombre de la aplicación integrada al modelo **IAM**.

`BASIC_AUTH`: Determina si se está utilizando Autenticación Básica. Valores permitidos: true o false.

`SERVICE_NAME`: Es el alias para el servicio de autenticación/autorización **IAM**. Salvo que el administrador cambie la configuración del servidor Web, el valor será IAMService.

Como ya se vió en la *Sección 4.4*, el tipo de sesión establecida con el modelo **IAM**, determina la estructura de la contraseña enviada a la función ***iam_authenticate*** de la API. Si se está utilizando *"passport"*, la estructura de la contraseña es la siguiente:

```
[ticketId][encryptedPassword]
```

En caso de no estar utilizando *"passport"*, sólo se envía la contraseña ingresada.

5.3.5 Herramienta de Administración

La herramienta de administración **IAM Admin**, es una aplicación Web muy sencilla, desde la cual se pueden realizar ciertas tareas de administración. Como ya se mencionó en el *Capítulo 4*, se definen dos tipos de usuario que pueden acceder a la aplicación. Estos dos tipos de usuario son: *Usuario Administrador* y *Usuario de Aplicación*.

IAM Admin, define estos dos tipos de usuario, mediante una clase particular denominada **UserRole**, que se puede apreciar en la Fig. 5.1. Cabe

destacar, que estos roles son definidos por **IAM Admin**, y son los que determinan su funcionamiento. Es por esto que la interfaz **UpdatableUser**, no maneja los roles como cadenas de caracteres, ni se deja a criterio de la implementación del módulo externo la definición de los mismos. Si se necesitan definir nuevos roles, debe extenderse la implementación de la herramienta de administración **IAM Admin** para que sean contemplados.

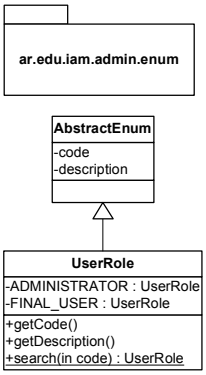

Figura 5.1: Clase **UserRole**

En la Fig. 5.2, se muestran los diferentes componentes que forman parte de la herramienta de administración **IAM Admin**.

5.3.6 Seguridad en la Comunicación

En esta sección se tratarán en detalle determinados aspectos de la comunicación con el modelo **IAM**, consideraciones de seguridad, tales como autenticidad de las partes que se comunican, y confidencialidad de la información que se envía por la red.

La seguridad es un punto crucial en los Servicios Web. Sin embargo, ni XML-RPC ni SOAP en sus especificaciones hacen explícitos ningún requerimiento de seguridad o autenticación.

- **Confidencialidad**: XML-RPC y SOAP corren principalmente sobre HTTP, y la comunicación XML puede entonces ser encriptada vía Secure Sockets Layer (SSL). SSL es una tecnología probada, y es una opción viable para encriptar los mensajes.

  La confidencialidad se logró a través del uso de certificados, como se vió en la *Sección 4.4*. En particular, se definieron dos tipos de certificados: el primero, es para obtener un canal seguro HTTPS (lo cual se logra configurando el Servidor Web con este certificado); y el segundo,

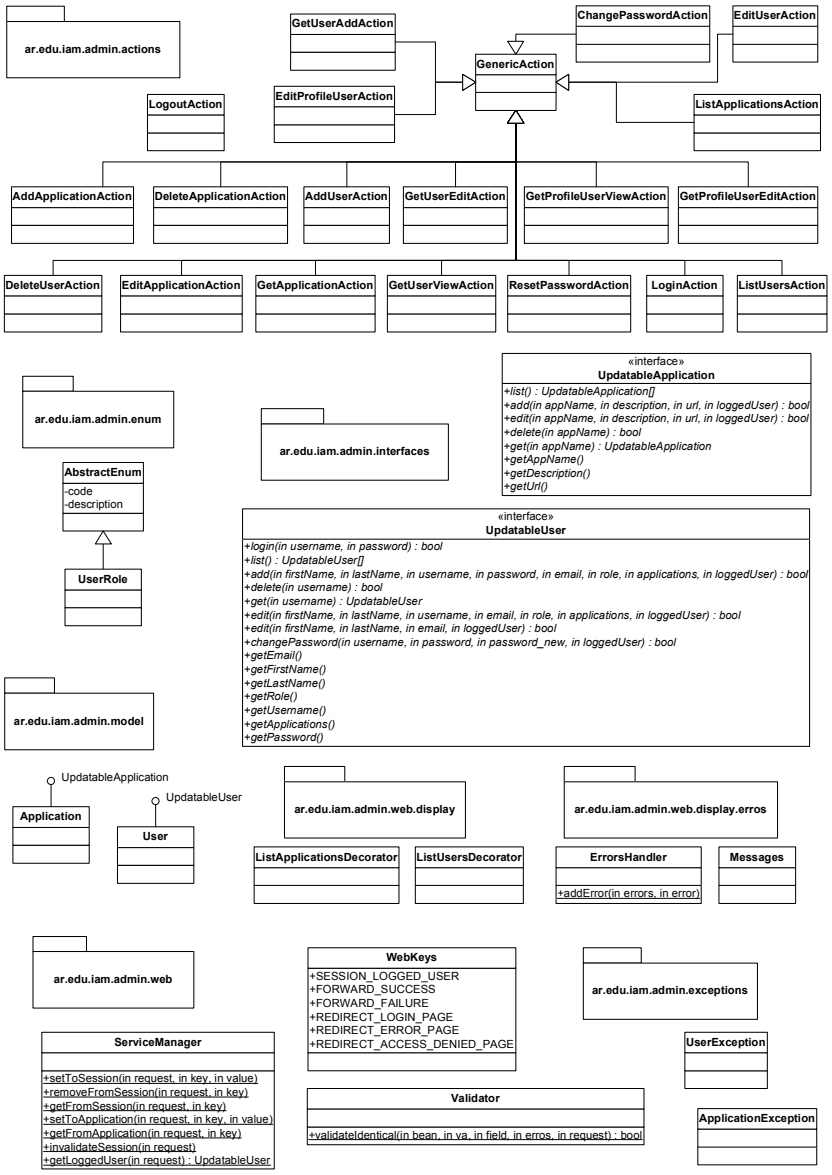

Figura 5.2: Diagrama de Clases - Herramienta de Administración **IAM Admin**

es para firmar digitalmente los clientes, de manera que éstos se puedan comunicar con el Servicio Web. Cabe destacar que se establece una política de autenticación mutua, determinando que sólo clientes autorizados se comuniquen con el Servicio Web y a su vez, que el Servicio Web sea una entidad de confianza para los clientes. Para la generación de certificados, se utilizó la herramienta **keytool**, y para firmar digitalmente los clientes, se utilizó **jarsigner**. Ambas herramientas son provistas por el SDK de Java.

- **Autenticación**: HTTP incluye soporte para Basic and Digest Authentication, y los servicios pueden entonces ser protegidos de la misma forma en que son protegidos los documentos HTML.

  La autenticación se logró entonces utilizando certificados con el agregado de Basic Authentication. Para implementar Basic Authentication, se cuenta con la siguiente configuración del framework Axis:

```
<login-config>
    <auth-method>BASIC</auth-method>
    <real-name>Protected Integrated Authentication Model</real-name>
</login-config>

<security-role>
    <description>IAM Security Role</description>
    <role-name>IAM</role-name>
</security-role>
```

  De esta manera, en el Servidor Web, se debe configurar el rol que puede acceder a este recurso protegido, y los usuarios que tengan este rol.

```
<user username="iam_basic_user" password="iam_pswd" roles="IAM"/>
```

  Los clientes del Servicio Web, se comunican utilizando el usuario y la contraseña vistas en el XML anterior.

- **Seguridad de Red**: extendiendo HTTP vía SOAP se faculta a los clientes remotos para invocar comandos y procedimientos, cosa que es prevenida por los firewalls. Si se quiere verdaderamente filtrar los mensajes XML-RPC o SOAP, se debe filtrar todos los requerimientos HTTP POST, PUT, DELETE, HEAD.

```
<security-constraint>
    <web-resource-collection>
```

```
        <web-resource-name>protected-resources</web-resource-name>
        <url-pattern>/services/IAMService</url-pattern>
        <http-method>HEAD</http-method>
        <http-method>POST</http-method>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>IAM</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

5.3.7 API integración módulo externo

Esta API se definió para permitir la comunicación entre los componentes **IAM Web Service** y **IAM Admin** del modelo **IAM** con las clases del módulo externo de autenticación como se muestra en la figura Fig. 5.3. El modelo interactúa con 4 clases:

- *User* implementa las interfaces **Authentication** y **Authorization**, para permitirle al servicio de autenticación/autorización **IAM** validar un usuario y determinar si tiene los permisos necesarios para acceder a la aplicación solicitada.
- *Cache* implementa la interfaz **Cacheable**, y permite administrar las sesiones de los usuarios.
- *User* implementa la interfaz **UpdatableUser**, e implementa la funcionalidad requerida por el módulo **IAM Admin** para el manejo de usuarios.
- *Application* implementa la interfaz **UpdatableApplication**, y permite administrar las aplicaciones del modelo.

Como ya se mencionó en el *Capítulo 4*, **IAM Web Service** es el encargado de definir las reglas del mecanismo de autenticación/autorización. Para esto se definieron tres interfaces que debe implementar el módulo externo para permitir que este proceso se lleve a cabo. Como ya se mencionó también, el mecanismo de autenticación consta de dos fases: la primera de **autenticación** y la segunda de **autorización**. Es por esto que se implementaron las interfaces **Authentication** y **Authorization**, que determinan respectivamente si un usuario es válido y si tiene los permisos para acceder a la

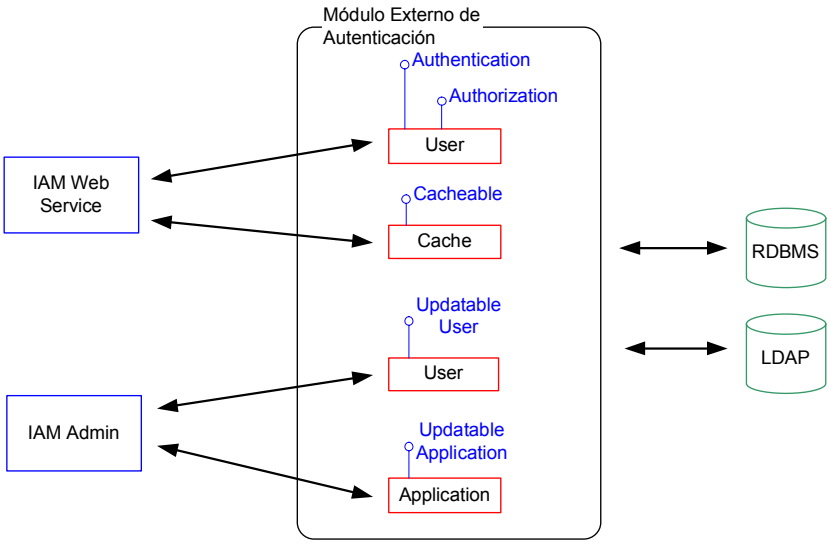

Figura 5.3: Implementación Módulo Externo según API Integración

aplicación solicitada. Una vez que el usuario ha sido validado y es correcto, a través de la interfaz **Cacheable**, se tiene la posibilidad de manejar la información de sesión generada.

A continuación se explica cual es el comportamiento de cada uno de los métodos de estas tres interfaces:

La interfaz **Cacheable** es la más compleja del modelo, y una de las más importantes. Las cuatro funciones siguientes, permiten acceder a la información de un ítem recuperado del cache.

getTicketId(): retorna el identificador unívoco de sesión generado por **SessionTicketGenerator** del modelo **IAM**.

getUsername(): retorna el nombre de usuario de una determinada sesión.

getSecretKey(): retorna la clave secreta de encriptación/desencriptación de una sesión determinada. Esta clave es generada por **SessionTicketGenerator** del modelo **IAM**.

getDate(): retorna la fecha de la sesión. A partir de esta fecha, el modelo **IAM** administra el tiempo de vida de las sesiones.

Los métodos que se explican a continuación, son para manipular la información de sesión, pudiendo recuperarla, almacenarla y modificarla.

storeCacheItem(long ticketId, byte[] key, String username): almacena la información de sesión. Esta función recibe el identificador unívoco de sesión *ticketId*, la clave de encriptación/desencriptación *key*, y el nombre de usuario *username* de la sesión. La sesión debe ser creada con la fecha actual de la operación. Ante cualquier error retorna la excepción **CacheException**.

getCacheItem(long ticketId): retorna la sesión especificada a través de *ticketId*.

getCacheItems(): retorna todas las sesiones almacenadas en el cache.

deleteCacheItem(long ticketId): elimina la sesión según *ticketId*. Esta función es utilizada por **CacheDeamon** del modelo **IAM**, para eliminar aquellas sesiones que han expirado, como así también cuando un usuario se desloguea.

updateCacheItem(long ticketId): actualiza la fecha de la sesión según *ticketId*. Esta función es utilizada para mantener activas las sesiones.

Las interfaces **Authentication** y **Authorization** son más simples. **Authentication** define el método isValid y **Authorization** havePermission.

isValid(String username, String password): retorna un booleano indicando si el usuario es válido a partir de su nombre de usuario *username* y su contraseña *password*. Cabe destacar que la contraseña que recibe está encriptada con la clave de encriptación/desencriptación única del modelo **IAM**. Propaga una excepción **AuthenticationException** ante cualquier problema o falla.

havePermission(String username, String application): determina si un usuario *username* tiene permiso para acceder a la aplicación solicitada *application*. Levanta una excepción **AuthorizationException** cuando se produce algún error.

Todos los posibles errores que puedan generar estos métodos son atrapados a través de las excepciones definidas para estas interfaces **AuthenticationException**, **AuthorizationException**, y **CacheException** según corresponda. El módulo externo de autenticación está obligado a manejar sus

propias excepciones y propagar hacia el modelo las excepciones antes mencionadas con la descripción de los errores ocurridos.

Ahora bien, hasta aquí se explicó qué comportamiento debe respetar e implementar el módulo externo para poder interactuar con el modelo **IAM**, y permitir realizar el mecanismo de autenticación/autorización **IAM**. Como ya se mencionó anteriormente, el modelo provee una herramienta para poder administrar los usuarios del modelo, permisos, y las aplicaciones que se integran. Para esto también, se definieron dos interfaces: **UpdatableUser** y **UpdatableApplication**. La primera permite manejar toda la información relativa a los usuarios del modelo, sus permisos, e información personal. La segunda permite administrar las aplicaciones, pudiendo agregar, eliminar y modificar aplicaciones.

Los métodos de la interfaz **UpdatableUser** y su comportamiento se ven a continuación:

getEmail(): retorna el e-mail del usuario.

getFirstName(): retorna el nombre del usuario.

getLastName(): retorna el apellido del usuario.

getRole(): retorna el rol del usuario. En el modelo **IAM** se contemplan dos tipos de roles, definidos a través de la clase **UserRole**: Usuario Administrador y Usuario de Aplicación. Un usuario con el rol de Administrador, podrá manejar la información de los usuarios, permisos y aplicaciones.

getUsername(): retorna el nombre de usuario.

getApplications(): retorna una lista con los nombres de las aplicaciones a las cuales el usuario puede acceder, es decir, que tiene los permisos suficientes para poder utilizarlas.

getPassword(): retorna la contraseña del usuario. Como ya se mencionó, esta contraseña está encriptada con la clave única de encriptación/desencriptación de **IAM**.

Estos métodos permiten acceder a la información de un usuario determinado.

login(String username, String password): retorna un booleano indicado si el proceso de login resultó exitoso. Este método recibe el nombre de usuario *username*, y la contraseña encriptada con la clave única de encriptación/desencriptación de **IAM** *password*, y a través de estos parámetros determina si el usuario existe y es válido.

list(): retorna una lista de todos los usuarios del modelo.

add(String firstName, String lastName, String username, String password, String email, UserRole role, String[] applications, String loggedUser): este método permite agregar un usuario al modelo. Retorna un booleano indicando si el proceso tuvo éxito o no. el parámetro *loggedUser*, representa el nombre del usuario logueado al sistema, para permitir realizar algún tipo de auditoría. El parámetro *applications*, representa el conjunto de aplicaciones a las cuales podrá acceder el usuario. La contraseña *password* está encriptada con la clave única de **IAM**. El parámetro *role*, es alguno de los definidos: Usuario Administrador o Usuario de Aplicación.

delete(String username): elimina un usuario del modelo según *username*.

get(String username): permite recuperar un usuario del modelo, a través del nombre de usuario *username*.

A continuación se explican dos métodos para editar usuarios. El primero es para que los usuarios con el rol de Administrador puedan modificar propiedades de los demás usuarios del modelo. El segundo, permite a los usuarios con el rol de Usuario de Aplicación, modificar sus opciones personales.

edit(String firstName, String lastName, String username, String email, UserRole role, String[] applications, String loggedUser): permite modificar un usuario según su nombre de usuario *username*. Los parámetros tienen el mismo sentido que el visto en el método **add**.

edit(String firstName, String lastName, String email, String loggedUser): permite modificar las opciones personales del usuario *loggedUser*. Este método es utilizado por los usuarios con el rol de Usuario de Aplicación.

El módulo externo debe contemplar cualquier excepción que se pueda producir, y devolver **UserException**.

La interfaz **UpdatableApplication** es más sencilla, y tiene los siguientes métodos que permiten administrar las aplicaciones del modelo **IAM**.

Estos métodos proveen una forma de acceder a los atributos de una aplicación:

getAppName(): retorna el nombre de una determinada aplicación.

getDescription(): retorna la descripción de una aplicación.

getUrl(): retorna la dirección (URL) de una aplicación.

Al igual que la interfaz **UpdatableUser**, **UpdatableApplication** tiene métodos para agregar, eliminar y modificar aplicaciones.

list(): retorna una lista de las aplicaciones integradas al modelo **IAM**.

add(String appName, String description, String url, String loggedUser): agrega una aplicación de nombre *appName*; *loggedUser* es usado para auditoría.

delete(String appName): elimina la aplicación de nombre *appName*.

get(String appName): retorna la aplicación de nombre *appName*.

edit(String appName, String description, String url, String loggedUser): permite editar la descripción *description* y la dirección (URL) de la aplicación de nombre *appName*; *loggedUser* es para auditoría.

Capítulo 6

Experiencias de Uso

Para mostrar el correcto funcionamiento del Integrated Authentication Model (**IAM**), se integran las siguientes aplicaciones:

- **ChikiWiki**: es una aplicación Web desarrollada en Java con Struts (MVC), que permite mantener una base de conocimiento común, pudiendo así compartir información y experiencias. Utiliza como fuente de datos una serie de archivos XML.

- **Mantis Bug Tracker**: es una aplicación Web desarrollada en PHP. Permite reportar y hacer el seguimiento de errores, pudiendo ver tipo de error, responsable de corrección y testeo, estado, etc. Almacena su información en una Base de Datos mySQL.

- **Jabber**: es una herramienta que brinda comunicación instantánea, desarrollada en C/C++. Es una aplicación cliente / servidor, en donde se puede agregar y remover contactos, enviarles mensajes, etc. Almacena la información en archivos XML.

Como se mencionó anteriormente, el modelo **IAM** depende de un módulo externo de autenticación, el cual debe respetar e implementar las interfaces mencionadas en el *Capítulo 4*, para almacenar y administrar la información necesaria para el proceso de autenticación/autorización, como es la información relativa a los usuarios, aplicaciones, permisos, y los datos de sesión.

Para este caso en particular, el módulo externo de autenticación utilizará una Base de Datos open source PostgreSQL [25] con la estructura que se ve en la figura Fig. 6.1.

En la tabla ***"iam_user"***, se guardará la información de los usuarios de la organización, como es su nombre y apellido, nombre de usuario, contraseña,

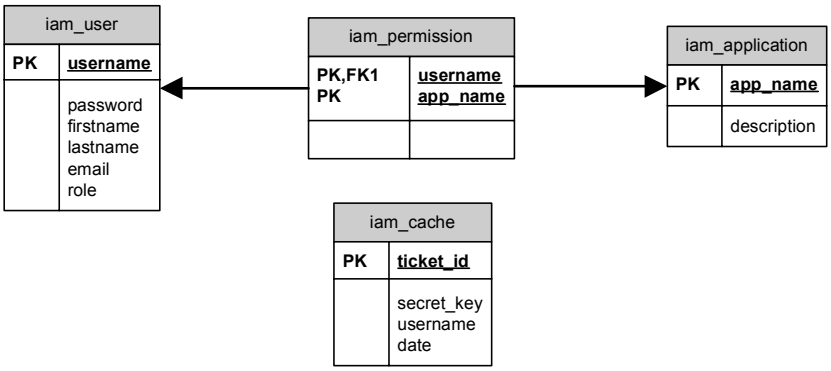

Figura 6.1: Diagrama de Entidad-Relación de la Base de Datos utilizada por el módulo externo de autenticación

e-mail y rol (el cual habilitará las distintas opciones de la herramienta de administración **IAM Admin**).

En la tabla ***"iam_application"***, estará la información de las aplicaciones disponibles (integradas) en el modelo, para que puedan ser utilizadas por los distintos usuarios.

La tabla ***"iam_permission"***, guardará los permisos de los usuarios sobre las aplicaciones.

En la tabla ***"iam_cache"*** estará la información sensible que maneja el modelo, permitiendo guardar los datos utilizados para realizar el proceso de revalidación de usuarios y manejo de sesiones.

Esta estructura básica permitirá saber si un usuario es válido en la organización, verificar su autenticidad, y saber si tiene los permisos suficientes para acceder a la aplicación que esta solicitando.

En las siguientes secciones, se verá cómo fueron integradas las tres aplicaciones mencionadas al inicio de esta sección utilizando las librerías de integración provistas en C++, PHP y Java.

6.1 ChikiWiki

Esta aplicación presenta una página de Login en la cual el usuario debe presentar un nombre de usuario y contraseña para ingresar como se ve en la Fig. 6.2.

En esta página, se embebió el Applet provisto por **IAM** (*Sección 5.2*), de manera que si se está usando passport, entonces el Applet automáticamente ingresará los datos dentro de los campos de la página, y submitirá el formulario de manera que se realice la autenticación con el modelo **IAM**.

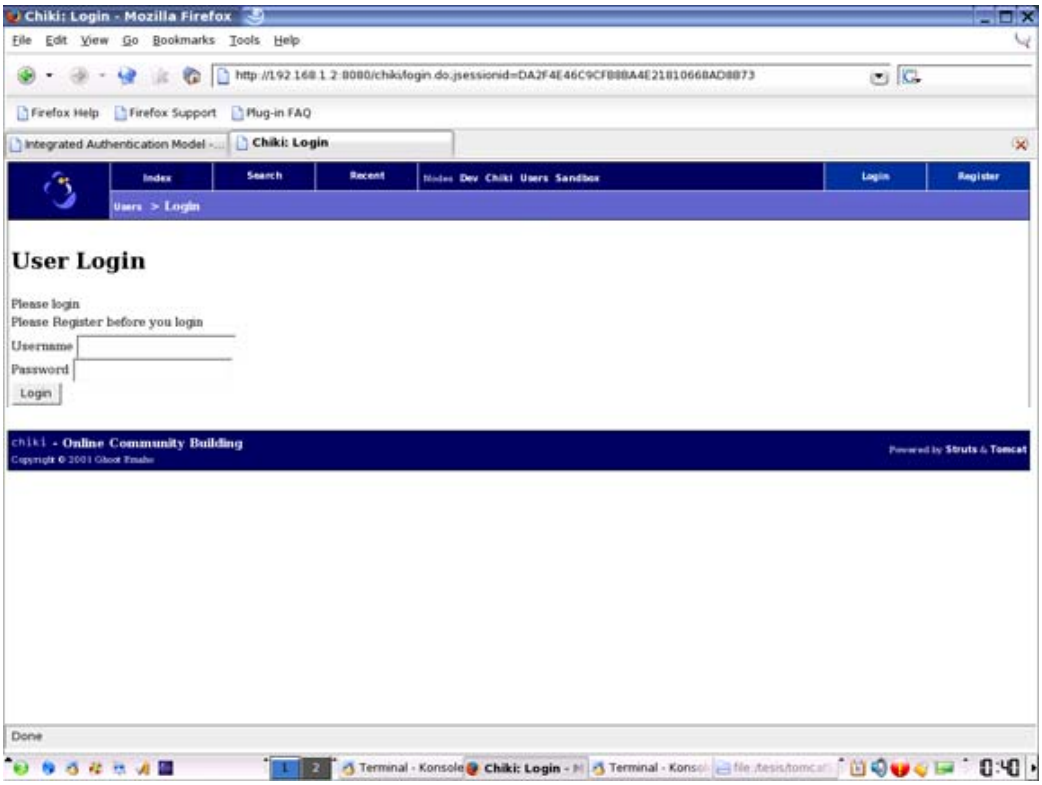

Figura 6.2: Página de Login de ChikiWiki

La submisión de este formulario, dispara una acción de Struts en el modelo de la ChikiWiki. Esta acción valida la información del formulario mediante su modelo propietario de autenticación. Es aquí donde hubo que cambiar esta validación por la llamada al servicio **IAM**. El código resultante, se puede ver en la figura Fig. 6.3

6.2 Mantis Bugtracker

La página inicial de esta aplicación, como se ve en la Fig. 6.4 nos da la posibilidad de ingresar el nombre de usuario y la constraseña para comenzar a utilizar la aplicación.

De manera similar a lo realizado en la ChikiWiki, se embebe el Applet en esta página inicial de manera que cuando se cargue la página, éste pueda verificar si ya existe una sesión de usuario establecida. De ser así, entonces el Applet automáticamente completa los dos campos de la página y submite el formuario.

La submisión del formulario, nos redirige a otra página PHP que es la

```
public User login(String requestedUsername, String requestedPassword)
    throws UnregisteredUsernameException, IncorrectPasswordException {

    if (usernameIsNotRegistered(requestedUsername))
        throw new UnregisteredUsernameException("No registration for specified username");

    User registeredUser = getUser(requestedUsername);
    int code = IAMAuthenticator.iam_authenticate(requestedUsername, requestedPassword);
    if (code != 0) {
       throw new IncorrectPasswordException("Password is incorrect");
    }

    return registeredUser;

} // end of login
```

Figura 6.3: Llamado al modelo **IAM** para realizar la autenticación

que realiza la validación del usuario. Esta página llama a una función, la cual decide, dependiendo del método de autenticación que tenga configurado, cómo valida la información recibida.

Mantis presenta un archivo de configuración, en el cual se puede establecer el método de autenticación. Tiene un conjunto de métodos soportados, entonces lo único que se hace es agregar otro y configurar para que lo use, como se muestra en la Fig. 6.5.

Luego se debe agregar la funcionalidad correspondiente al nuevo método de autenticación en la función *auth_does_password_match*. El código resultante, se puede ver en la Fig. 6.6

6.3 Jabber

Jabber no es una aplicación en sí, sino que es un protocolo para mensajería instantánea. Lo que se integró en realidad son un cliente y un servidor que soportan este protocolo de comunicación.

Como cliente, se eligió Gaim [26], que es un cliente open-source multiplataforma con soporte para Jabber, ICQ, MSN, etc. Como servidor Jabber, se eligió jabberd 1.4 [27], que también es multiplataforma y open-source.

Al inicio, Gaim presenta al usuario la pantalla que se ve en la Fig. 6.7.

Para el uso de *"passport"*, hubo que modificar el comportamiento de la conexión, para que las cuentas de Jabber verifiquen el uso del mismo, y si ya tienen establecida una sesión de usuario con el servicio **IAM**, no se le requiera al usuario el nombre de usuario y contraseña para utilizarlo. El resultado de esta modificación, se puede ver en la Fig. 6.8.

También en la Fig. 6.8 se puede apreciar que Gaim no valida directamente contra el servicio **IAM**, sino que setea la contraseña con el resultado de la

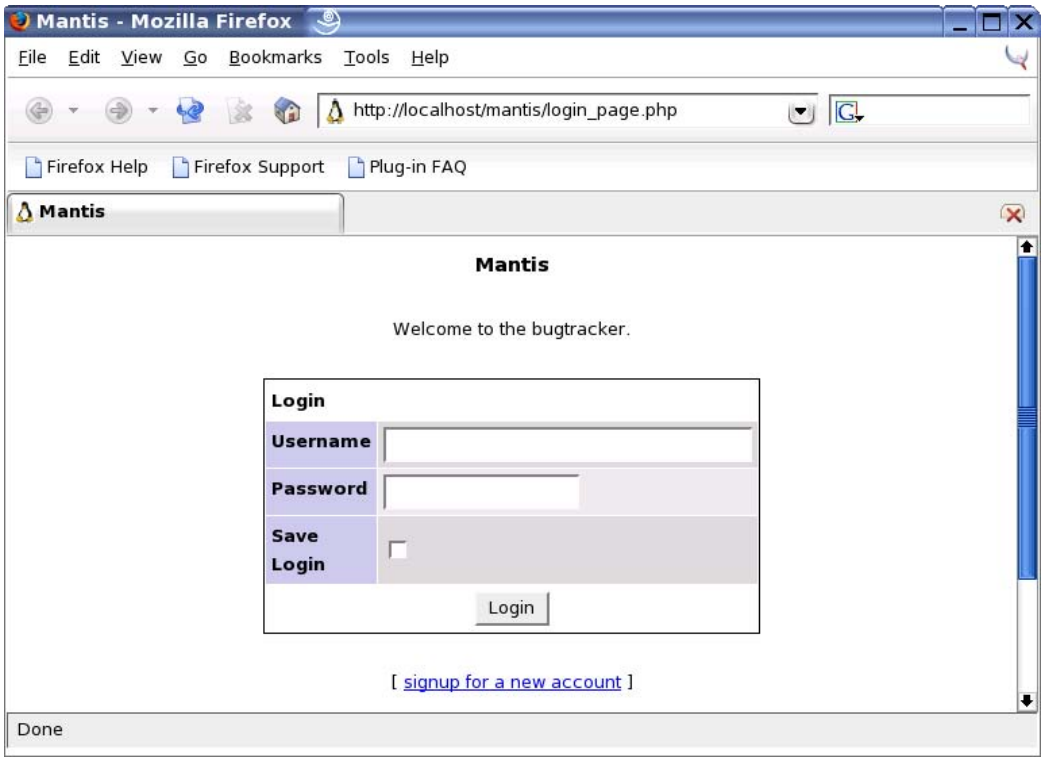

Figura 6.4: Página de Login de Mantis

función provista por la librería de integración, para enviársela al servidor Jabber y así indicarle que el usuario está utilizando passport y tiene iniciada una sesión de usuario.

En caso de no estar utilizando *"passport"*, se le pedirá al usuario que ingrese su nombre y contraseña de manera normal.

El servidor Jabber, está preparado para la integración de nuevos métodos de autenticación. Esto se lleva a cabo desarrollando un módulo en Perl, el cual se invoca cuando se está intentando autenticar un usuario. Dentro de éste módulo, se puede implementar cualquier método de autenticación. En particular, puede implementarse el llamado al servicio **IAM**.

Para esto, se utilizó el paquete Inline [28], que permite utilizar otros lenguajes dentro de Perl. Entonces, se incluyeron las librerías de integración hechas en C para comunicarse con el servicio **IAM**. El resultado de la implementación del módulo en Perl se ve en la Fig. 6.9.

```
# --- login method ----------------
# CRYPT or PLAIN or MD5 or LDAP or BASIC_AUTH or IAM
$g_login_method = IAM;
```

Figura 6.5: Parte del archivo de configuración de Mantis, donde se ve el cambio realizado para integrar con el modelo **IAM**

```
function auth_does_password_match( $p_user_id, $p_test_password ) {
        $t_configured_login_method = config_get( 'login_method' );

        if ( LDAP == $t_configured_login_method ) {
                return ldap_authenticate( $p_user_id, $p_test_password );
        }

        if ( IAM == $t_configured_login_method ) {
                $s_username = user_get_field( $p_user_id, 'username');
                return ( iam_authenticate( $s_username, $p_test_password ) == 0 );
        }
```

Figura 6.6: Llamado al modelo **IAM** para realizar la autenticación desde la función *auth_does_password_match* de Mantis

Figura 6.7: Pantalla inicial de Gaim

```
if (strcmp(gaim_account_get_protocol_name(account), "Jabber") == 0) {
   //For jabber protocol, check if using IAM as authentication and if passport is enabled
   int cup = iam_can_use_passport();
   if ( cup ) {
      struct iam_passport info;
      iam_get_passport_info(&info);
      gaim_account_set_password(account,iam_ensemble_password(&info));
   }
}
```

Figura 6.8: Modificación realizada en Gaim para el uso de *"passport"*

```
use Inline (C => Config =>
        ENABLE => AUTOWRAP =>
        LIBS => "-L/home/fer/jabber/iam -liam -lssl -lcrypto" =>
        LDDLFLAGS => "-L/home/fer/jabber/iam -liam -lssl -lcrypto -shared",
);
use Inline C => <<'END_C';
   #include <iam/iam.h>
   int iam_authenticate (char* username, char* password);

   int authenticate(char * user, char * password) {
      int retVal = iam_authenticate(user, password);
      return retVal;
   }
END_C

sub auth_check
{
  my($user, $pass) = @_;

  if (authenticate($user, $pass) == 0) {
    # Auth success
    return(0);
  }
  else
  {
    # Auth failure
    return("Invalid password");
  }
}
```

Figura 6.9: Módulo en Perl utilizado para utilizar el modelo **IAM** dentro del servidor jabberd

Capítulo 7

Conclusiones

En esta sección se verá cuáles de los requerimientos vistos en el *Capítulo 2* fueron contemplados, y en qué medida fueron satisfechos. Además se mostrarán algunos trabajos futuros que se pueden realizar para mejorar la implementación del modelo **IAM**.

El modelo **IAM**, desde su concepción, fue diseñado de tal forma que se pudiera adaptar a los distintos escenarios con un costo relativamente bajo. Debido al alto nivel de abstracción, sumado con la utilización de las tecnologías más convenientes para su implementación, se logró un modelo, que no sólo contempla todos los requerimientos planteados, si no, que además tiene un valor agregado con respecto a las soluciones de la actualidad que resuelven la problemática planteada, vistas en el *Capítulo 3*.

Uno de los puntos mas fuertes en el diseño del modelo **IAM**, es que permite integrar aplicaciones que responden a distintas arquitecturas, provistas por distintas organizaciones, e implementadas con diversos lenguajes, con un bajo costo, haciendo casi transparente la utilización del modelo para el usuario de las aplicaciones, como así también para el administrador.

Otro punto para destacar del modelo, es que brinda un esquema de autorización, que no es soportado por las demás soluciones. Es decir, con las soluciones vistas en el *Capítulo 3*, al integrar aplicaciones, estas quedan disponibles para todos los usuarios, y esto representa una dificultad, dado que no es el escenario que se quiere dentro de una organización, debido a que no todos los usuarios tienen los mismos roles, ni realizan las mismas tareas. El modelo **IAM**, permite asignar permisos sobre las aplicaciones integradas, de forma de proveer una esquema de autorización flexible, que se adapte a las necesidades de la organización.

Otra de las ventajas del modelo, es que provee una herramienta para administrar la información de los usuarios, permisos y aplicaciones, posibilitándole de esta forma, una manera ágil y sencilla al administrador, de integrar nuevas aplicaciones, agregar nuevos usuarios, y manejar los permisos de acceso.

Además de todo lo expuesto anteriormente, cabe destacar que **IAM**, es una solución open-source, con un diseño orientado a objetos, modular, y extensible, con muy pocas líneas de código, lo que lo convierten en una solución muy fácil de entender, y extender o modificar su comportamiento o funcionalidad. Como valor agregado, está desarrollado en el lenguaje Java, que es uno de los más populares de la actualidad, y del cual se puede encontrar mucha documentación y ejemplos.

Hasta aquí se nombraron algunas de las ventajas del modelo **IAM** con respecto a las demás soluciones de la actualidad. Uno de los puntos quizás más débiles, es el de la integración con los módulos de autenticación externos, dado que hay que implementar esta integración. De todos modos, las interfaces provistas, como ya se vió en la *Sección 5.3.7*, permiten un manejo flexible, y a su vez, dan una guía de como realizar esta implementación, dado que proveen la documentación correspondiente para determinar qué hacer en cada caso. En este punto, se plantea como trabajo futuro, diseñar algún tipo de framework que permita esta integración de una forma más ágil y transparente.

En la Fig. 7.1 se muestra una tabla comparativa entre las tres aplicaciones ya vistas en el *Capítulo 3* y el modelo **IAM**, en relación a los requerimientos planteados en el *Capítulo 2*.

7.1 Trabajo Futuro

Aquí se plantean algunas mejoras que se podrían implementar para extender la funcionalidad, optimizar su rendimiento, y posibilitar una integración más sencilla dentro de una organización.

- Implementar un "Pluggable Framework", de forma tal de poder integrar **IAM** con cualquier sistema externo de autenticación, como por ejemplo LDAP y Active Directory.
- Mejorar la documentación del modelo.

	CAS	JOSSO	PubCookie	IAM
Fácil integración	✓	✓	✓	✓
Mecanismo de autenticación múltiple	✓	✓	✓	✓
Mecanismo de autorización				✓
Independencia de las arquitecturas				✓
Manejo de sesión única	✓	✓	✓	✓
Centralización de la información	✓	✓	✓	✓
Administración de la información				✓

Figura 7.1: Tabla comparativa de las aplicaciones CAS, JOSSO, PubCookie, **IAM**

- Agregar mayor funcionalidad a la herramienta de administración **IAM Admin**, como por ejemplo, recuperación de contraseñas y envío de e-mails.

- Utilizar SOAP sobre BEEP (Blocks Extensible Exchange Protocol) en lugar de SOAP sobre HTTP. BEEP es un framework para construir nuevos protocolos. En particular, BEEP trabaja sobre TCP e incluye ciertas características built-in, incluyendo un protocolo handshake, autenticación, seguridad y manejo de errores.

Bibliografía

[1] G. Henri ter Hofte.
"Working apart together : Foundations for component groupware"
Telematica Instituut - 1998

[2] Jenny Preece.
"On Line Communities" Designing Usability, Supporting Sociability.
Ed. Wiley.- 2000

[3] Definición de Mensajería Instantánea.
http://www.sharpened.net/glossary/definition.php?IM

[4] Bo Leuf - Ward Cunningham.
"The Wiki Way"
Ed. Addison-Wesley Longman - 2001

[5] Definición de "Bug Tracker".
http://en.wikipedia.org/wiki/Bug_Tracker

[6] CAS (Central Authentication System).
http://www.yale.edu/tp/auth/

[7] JOSSO (Java Open Single Sign-On).
http://www.josso.org/

[8] PubCookie.
http://www.pubcookie.org/

[9] Data Encryption Standard.
Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C.

[10] Java technology.
http://java.sun.com/

[11] SOAP Specifications.
http://www.w3.org/TR/soap/

[12] Apache. Axis Documentation.
http://ws.apache.org/axis/

[13] Apache. Log4j Documentation.
http://logging.apache.org/log4j/

[14] Applets.
http://java.sun.com/applets/

[15] Java Web Start technology.
http://java.sun.com/products/javawebstart/

[16] Apache Struts Web Application Framework.
http://struts.apache.org/

[17] Java Servlet Technology.
http://java.sun.com/products/servlet/

[18] JDBC Technology.
http://java.sun.com/products/jdbc/

[19] JavaServer Pages Technology.
http://java.sun.com/products/jsp/

[20] JavaServer Pages Standard Tag Library.
http://java.sun.com/products/jsp/jstl/

[21] DisplayTag Library.
http://displaytag.sourceforge.net/

[22] gSOAP. C/C++ Web Services and Clients.
http://www.cs.fsu.edu/ engelen/soap.html

[23] Dietrich Ayala. NuSoap.
http://dietrich.ganx4.com/nusoap/index.php

[24] Gamma et al. “Singleton” Pattern. 1995

[25] PostgreSQL Database.
http://www.postgresql.org/

[26] Gaim. A multi-protocol instant messaging (IM) client.
http://gaim.sourceforge.net/

[27] jabberd project.
http://jabberd.jabberstudio.org/

[28] Inline.
http://inline.perl.org/