UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

INGENIERÍA SUPERIOR DE TELECOMUNICACIONES

PROYECTO FIN DE CARRERA

Integración de un Sistema de Análisis de Contenidos para una Plataforma de Interceptación Legal de Comunicaciones

AUTOR: RAQUEL APARICIO MORENILLA
TUTOR: MANUEL URUEÑA PASCUAL

Resumen

La presente memoria documenta el proceso de integración de varios módulos y herramientas con el objetivo de obtener una herramienta de análisis de contenidos que permita agilizar las labores de los agentes de la ley encargados de analizar la información intercambiada por un sospechoso. Dicha información habrá sido obtenida mediante una interceptación legal, partiendo de la base de que un juez accede a que se intercepten las comunicaciones de un sospechoso, y firma una autorización legal que permita realizarla. Partiendo de una herramienta forense ya existente, *Xplico*, que permite obtener los mensajes de nivel de aplicación a partir de un fichero de captura de tráfico, se ha incorporado la funcionalidad de preclasificación de los mismos, mediante llamadas a un módulo de distribución de contenidos que, según el tipo del contenido, escoge un subconjunto de los denominados *plugins* de preclasificación, que obtienen su relevancia en el caso. Para mayor información, también se proporciona un comentario textual que incluye más datos acerca del procesamiento. Estos *plugins* permiten ser entrenados para mejorar el funcionamiento de la herramienta. Por otro lado, también se ha integrado un módulo de transcripción VoIP que automáticamente transcribe contenidos de audio obtenidos de conversaciones realizadas mediante SIP, de manera que el texto de las transcripciones pueda ser procesado por el analizador de contenidos. También es posible entrenar este módulo y mejorar así la precisión de la transcripción. La mayoría de las herramientas integradas: módulo distribuidor, comunicación con *plugins* y módulo de transcripción habían sido desarrollados previamente pero han sido necesarias profundas modificaciones para realizar la integración. Por último, se ha mejorado la interfaz web de la herramienta original que permite acceder a los contenidos, de manera que ahora el control de acceso es realizado mediante certificados digitales almacenados en tarjetas inteligentes, y directorios LDAP y permite entrenar tanto el módulo de transcripción como los diferentes *plugins* y configurar el módulo distribuidor de contenidos mediante la adición, edición y eliminación de *plugins*. Es también a través de esta interfaz donde se ha realizado la integración con LINK, una herramienta que por ejemplo ofrece al agente una visualización del flujo de correos o llamadas VoIP.

Abstract

This document summarizes the process of integration of several modules and tools with the aim of obtaining a single content preclassifying tool to assist Law Enforcement Agents in charge of analyzing the exchange of information performed by a suspect. This information can only be obtained when a Lawful Authorization has been granted by a Judge who allows the suspect's communication to be intercepted and therefore signs the required Lawful Authorization. The process starts from a traffic capture file containing the subject's communications, which is analyzed by the forensic tool *Xplico* which obtains the application messages in it, which will be referred to as contents. *Xplico* has been modified to perform calls to a content distribution module which, depending on the type of content, delivers them to different sets of small applications, called *plugins*. These sets of *plugins* provide the relevance for the content in a certain case and offer a training mode to improve their performance. Furthermore, a VoIP transcription module has also been integrated into *Xplico*. This module automatically transcripts audio contents from SIP conversations so that transcriptions can be processed by the content analyzer. Moreover, these transcripted conversations can also be employed to train and improve the performance of the module. All these modules: content distribution module, communication to *plugins* and transcription module had been previously developed but some modifications have been done in order to achieve the integration. Last but not least, the web interface which gives access to the contents has been improved so that access control is performed by means of digital certificates stored in smart tokens and LDAP directories. Besides, it now offers the possibility of training both the transcription module and the *plugins* and of configuring content distribution module through the addition, edition and deletion of *plugins*. It is also through this interface where integration with a tool called LINK has been performed, which for example allows agents to visualize the flow of emails or VoIP calls in the form of graphs.

Contenido

Índice de figuras

Glosario

A continuación se proporciona una serie de abreviaturas frecuentemente utilizadas en este documento y que concuerdan con las utilizadas en los estándares consultados. En el siguiente apartado se puede consultar una definición completa de las mismas.

AP: proveedor de acceso (*access provider*).

CA: autoridad de certificación (*certification authority*).

CC: contenido de la comunicación (*content of communication*).

DWW: autorización digital de escucha (*Digital Wiretap Warrant*).

ETSI: Instituto Europeo de Estándares de Telecomunicaciones (*European Telecommunications Standards Institute*).

HI: interfaz de entrega (*handover interface*).

ILIP: plataforma de intercepción legal de comunicaciones de INDECT (*INDECT Lawful interception platform*).

INDECT: sistema inteligente de información que soporta observación, búsqueda y detección para la seguridad de los ciudadanos en entornos urbanos (*Intelligent information system supporting observation, searching and detection for security of citizens in urban environment).*

IRI: información relacionada con la interceptación (*intercept related information*).

LEA: agencia encargada del cumplimiento de la ley (*law enforcement agency*).

LEMF: instalación de monitorización de la LEA (*law enforcement monitoring facility*).

LI: interceptación legal (*lawful interception*).

NWO: operador de red (*network operator*).

PKI: Infraestructura de clave pública (*Public Key Infrastructure*).

SvP: proveedor de servicios (*service provider*).

VoIP: voz sobre IP (*voice over IP*).

Definiciones

Agencia encargada del cumplimiento de la ley (*law enforcement agency o LEA*): Servicio autorizado por ley para llevar a cabo la interceptación de las comunicaciones.

Autorización legal (*wiretap warrant)*: Permiso dado a un LEA bajo determinadas condiciones para interceptar unas telecomunicaciones concretas. Normalmente se refiere a una orden proporcionada por un cuerpo legalmente autorizado.

Contenido de la comunicación (*content of communication o CC*): Información intercambiada entre usuarios de un servicio de comunicaciones, incluyendo información almacenada para posteriormente ser accedida y que excluye la IRI.

Identidad: Etiqueta técnica (normalmente un número) que representa el origen o destino de cualquier tráfico de telecomunicaciones.

Identidad objetivo: Identidad correspondiente al sujeto bajo investigación. Normalmente un sujeto tendrá varias identidades objetivo.

Información relacionada con la interceptación (*intercept related information* o IRI): Información asociada con los servicios de comunicación que se refiere a la identidad del sujeto bajo investigación, tal como información de comunicación (intentos fallidos de establecer una comunicación), de servicios asociados (gestión del perfil del servicio) o de localización.

Instalación de monitorización para el cumplimiento de la ley (*law enforcement monitoring facility* o LEMF): Instalación de un LEA a las que las operadoras deben enviar los resultados de la interceptación de un determinado sujeto

Interceptación legal (*lawful interception* o LI): Acción regulada por ley realizada por los NWO, AP o SvP de proveer las LEAs con las telecomunicaciones e información relacionada de los sujetos bajo investigación.

Interfaz de entrega (*handover interface* o HI): Interfaz física y lógica que transmite las peticiones de interceptación y los resultados de la misma entre los NWO, AP y SvP y las LEAs. Consiste en tres puertos: HI1 para información de administración, HI2 para la IRI y HI3 para los CC)

Operador de red (*network operator* o NWO): Operador de una infraestructura de telecomunicaciones.

Proveedor de acceso (*access provider* o AP): Proporciona a un usuario de una red acceso a la misma. Puede corresponder o no con el operador de dicha red.

Proveedor de servicios (*service provider* o SvP): Proporciona servicios de telecomunicaciones sobre una red, propia o no.

Sujeto bajo investigación: Persona identificadas en la autorización legal cuyas telecomunicaciones tanto recibidas como enviadas van a ser interceptadas y monitorizadas.

Capitulo 1: **Introducción**

El presente proyecto fin de carrera se enmarca dentro del proyecto de investigación INDECT [1] cuyo objetivo es el desarrollo de herramientas tanto para entornos reales como virtuales que mejoren la seguridad de los ciudadanos y protejan la confidencialidad de la información grabada y almacenada, así como la privacidad de las personas relacionadas con dicha información. Este proyecto europeo está organizado en varios paquetes de trabajo y la Universidad Carlos III de Madrid está presente en varios de ellos. Concretamente, este proyecto fin de carrera y el estudio tecnológico relacionado con él forman parte del trabajo que se está realizando en el tercer paquete de trabajo, denominado WP3, que consiste en el desarrollo de una plataforma de interceptación legal de comunicaciones (*Lawful Interception* o LI), como la mostrada en la Figura 1. La función de dicha plataforma es tanto capturar las comunicaciones de los sujetos bajo investigación, así como agilizar en lo posible algunas tareas de las agencias encargadas del cumplimiento de la ley (*Law Enforcement Agency* o LEA) mediante una preclasificación automática de los contenidos interceptados. Usamos el término preclasificación puesto que ha de ser supervisada y revisada por analistas autorizados que decidirán la clasificación final. Tanto la automatización en sí como la preclasificación obtenida permiten por un lado organizar y aprovechar los recursos de una forma mucho más eficiente y por otro disminuir la intervención humana en el proceso, lo que a su vez disminuye la subjetividad, los errores y la posibilidad de que la información sea usada con fines ilegítimos o se produzcan violaciones de los derechos individuales.

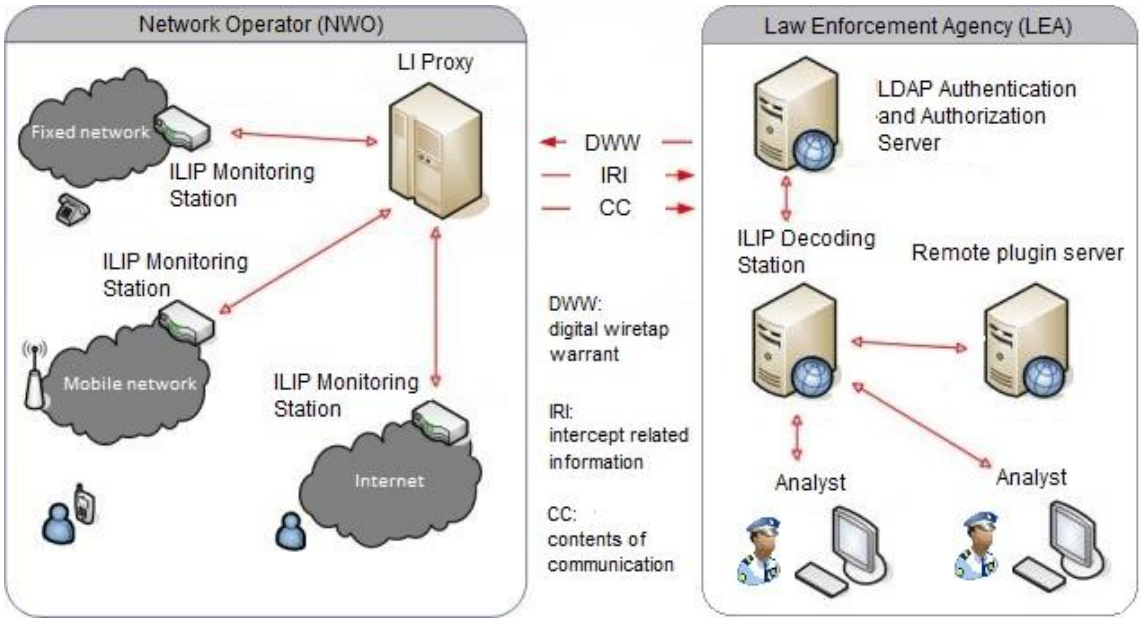


Figura 1. Esquema de Interceptación Legal de Comunicaciones.

Por simplicidad nos refereriremos a la plataforma como ILIP: *INDECT Lawful Interception Platform*. Ésta está formada por dos sistemas. El primero es una estación de captura de información capaz de identificar la información correspondiente a un determinado sujeto bajo investigación y filtrar todo el tráfico no relacionado. Esta estación se sitúa en un lugar en el que tenga acceso al flujo de datos, esto es, dentro de la red de un operador de telecomunicaciones, que se corresponde con la parte izquierda

de la Figura 1. La segunda es una estación decodificadora que se encarga de analizar dicha información y realizar una preclasificación de los contenidos intercambiados. Esta última se sitúa en las instalaciones de las LEAs, correspondiente a la parte derecha de la Figura 1.

Además, se ha definido un mecanismo criptográfico denominado autorización legal digital (*Digital Wiretap Warrant* o DWW) que asegura la legalidad del proceso y la protección y veracidad de los datos, aunque éste no se explicará en detalle puesto que queda fuera del ámbito de este Proyecto Fin de Carrera. Este mecanismo se aplica a todo el flujo de información generado en el proceso, tanto internamente entre los distintos elementos involucrados en la captura de información, su procesado y su entrega, como en la comunicación entre los dos sistemas que componen ILIP.

1.1 Motivaciones

La interceptación es la acción de apoderarse de algo antes de que llegue a su destino. Sin embargo, en el caso de las comunicaciones, se trata de obtener una copia de la información enviada o recibida de forma que la parte o partes que están siendo interceptadas no tengan conocimiento de ello. Esto supone una clara invasión del derecho a la intimidad pero a su vez tiene un papel fundamental para combatir las actividades criminales. En este contexto encontramos dos posibles situaciones: que ya se ha haya producido un crimen y las comunicaciones de un sujeto puedan aportar información o constituir una prueba judicial, o que haya indicios de que se vaya a producir un crimen y el acceso a la información de determinados sujetos pueda aportar información para evitarla. En cualquiera de los dos casos es necesario el establecimiento de una regulación clara al respecto, de manera que no sea posible que se produzcan interceptaciones sin autorizaciones judiciales, previa presentación de pruebas o indicios que las justifique. Dichas autorizaciones son básicamente órdenes legales que especifican los detalles concretos para una determinada interceptación: sujeto, condiciones y resultados, y que dependen de las legislaciones nacionales. Esto nos lleva a la denominada interceptación legal que es la interceptación de las comunicaciones de un sujeto aprobada legalmente y que es la base de este proyecto.

Vemos que en un primer momento se ha hablado de comunicaciones en general; sin embargo al hablar de interceptación legal nos referiremos a las telecomunicaciones. La diferencia esencial consiste en que mientras que el concepto de comunicación representa el intercambio de mensajes entre individuos, el de telecomunicación engloba la técnica: radio, telegrafía, televisión, telefonía, transmisión de datos e interconexión de computadoras. Esto implica que el proceso de interceptación depende del tipo de tecnología utilizado. Por ejemplo, en las redes telefónicas tradicionales se establecía y dedicaba un circuito de voz para cada conversación y además los terminales eran muy sencillos, por lo que resultaba fácil interceptar la comunicación en algún punto. Hoy en día, la proliferación de comunicaciones sobre redes basadas en la tecnología de protocolo internet (*Internet Protocol* o IP) hace necesario vigilar el acceso a Internet y los servicios basados en IP, principalmente el correo electrónico, los protocolos de voz por Internet (*Voice Over IP* o VoIP) pero también otros como los sistemas de información basados en web, el comercio electrónico o la difusión de video. Todo este tráfico debe ser interceptado y analizado para posteriormente poder ser utilizado en las investigaciones judiciales. Sin embargo, esta tarea no es fácil ya que el tratamiento de este tráfico de Internet añade complicaciones:

- La identidad de los participantes en un flujo de información está embebida en dicho flujo.
- Es posible que se mezclen los datos del sujeto investigado con datos de otros individuos, no sólo los otros interlocutores en el caso de una conversación, por lo que todo el tráfico no relacionado con la comunicación del sujeto bajo investigación debe ser filtrado.
- Suele haber varias partes participando en el transporte de los datos, tales como un proveedor de acceso, un operador de red y uno o varios proveedores de servicios. Es común que la infraestructura del proveedor de acceso pertenezca al operador de red pero no tiene porqué ocurrir siempre así.
- En muchos países la regulación referente a cómo gestionar la interceptación legal de datos no está clara, lo que permite que un proveedor de servicios pueda eludirlas puesto que implica un coste adicional para el mismo.
- La separación del flujo de datos relevante del resto de flujos de información implica un esfuerzo que incluye desarrollo software y recursos de computación.

En la Figura 2 se observa la relación entre los tres elementos por los que circula el tráfico de red. Veamos cuál es la función de cada uno:

- Un operador de red (*Network Operator* o NWO) es la empresa que gestiona las líneas de comunicación. En España encontramos entre otras ONO, Orange, Telefónica y Vodafone, que forman parte de la asociación de operadores de telecomunicaciones con red propia Redtel [2].
- Un proveedor de acceso (*Access Provider* o AP) es la compañía que ofrece el acceso a internet a través de distintas tecnologías como pueden ser DSL, fibra óptica o de forma inalámbrica. También se le denomina proveedor de servicios de Internet (*Internet Service Provider* o ISP). Puede tratarse del propio operador de red. En España encontramos Jazztel, Movistar (red de Telefónica) u ONO por ejemplo.
- Un proveedor de servicios (*Service Provider* o SvP) ofrece algún tipo de servicios a través de la red: servicio de correo electrónico, servicio de alojamiento de páginas web, servicio de aplicación (*Application Service Provider* o ASP), servicio de comercio electrónico…etc, como por ejemplo Hotmail, Skype, Google (Gmail, GTalk, Google Maps,…), Amazon Web Services o PayPal, por mencionar algunos ejemplos variados.

El cliente necesita tener un equipamiento denominado equipo local de cliente (*Customer Premises Equipment* o CPE) que se configura para comunicarse con el proveedor de acceso y obtener así conexión a una red, ya sea Internet o una Intranet. La Figura 2 muestra un operador que proporciona el acceso a su propia red, pero como ya se ha dicho, puede ser otra compañía la que lo haga. Por último, observamos los proveedores de servicio, a los que se tiene acceso a través de dicha red, que son típicamente a los que el usuario quiere conectarse para comprobar su correo, chatear o mantener conversaciones de VoIP y visitar o mantener sitios web.

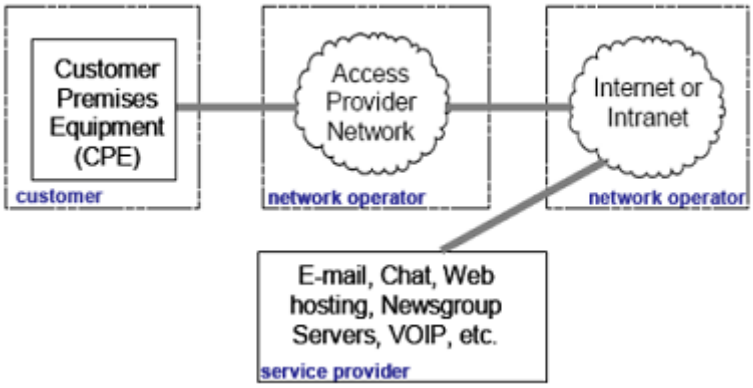


Figura 2. Proveedores de acceso, operadores de red y proveedores de servicio [1].

Vistas estas complicaciones y dado que cada operador conoce la estructura y tiene acceso directo a los flujos de información que pasan por él, y que el acceso de una LEA a dicha información debe estar limitado para evitar la posibilidad de interceptaciones ilegales o falsificación de datos, en este proyecto se parte de la base de que la interceptación es realizada por los operadores. Hay que reiterar que sólo se deben realizar interceptaciones y entregar información a un LEA previa entrega de la correspondiente autorización judicial, para lo cual será necesario aplicar medidas de control tanto preventivas como para descubrir si se ha producido algún uso indebido. Bajo estos requisitos se distinguen dos situaciones básicas para los operadores: que se les requiera almacenar la información intercambiada por sus clientes por un tiempo determinado (durante el cual dicha información puede ser solicitada), o que se les requiera realizar interceptaciones a un determinado sujeto a petición y enviar dicha información a la LEA lo más rápido posible. El primer caso no se considera interceptación en sí pero evidencia la necesidad de almacenar y discernir la información correspondiente a los usuarios. En ambos casos es necesario un mecanismo para entregar los datos a las autoridades de manera segura, que garantice la protección e integridad de los datos.

Puesto que este proyecto se encuadra en el contexto de la interceptación legal de comunicaciones, aparte del desarrollo de la plataforma en sí, se ha considerado imprescindible investigar la situación actual de la misma y los estándares existentes. Debido a que el proceso de interceptación depende de la estructura de cada operador, los estándares que se han desarrollado en torno a la LI principalmente intentan facilitar la cooperación entre los operadores y las LEAs, aunque también existe una gran cantidad de estándares dedicados específicamente a diferentes tipos de redes y a diferentes servicios de comunicaciones. El estudio realizado se centra en los estándares del Instituto Europeo de Estándares en Telecomunicaciones (*European Telecommunications Standards Institute* o ETSI), tanto en los que definen la arquitectura general como en los que se aplican a las redes de paquetes IP, ya que este es el caso que nos atañe dentro del proyecto INDECT, del que hablaremos a continuación.

1.2 Objetivos

El proyecto INDECT es un proyecto de investigación del séptimo programa Marco de la Unión Europea que comenzó en enero de 2009 con una duración prevista de 60 meses y en el que participan 11 Universidades, 4 empresas y dos usuarios finales, el servicio de policía de Irlanda del Norte y el Ministerio del Interior de Polonia. Su objetivo es el desarrollo de herramientas tanto para entornos reales como virtuales que mejoren la seguridad de los ciudadanos y protejan la confidencialidad de la información grabada y almacenada, así como la privacidad de las personas relacionadas con dicha información.

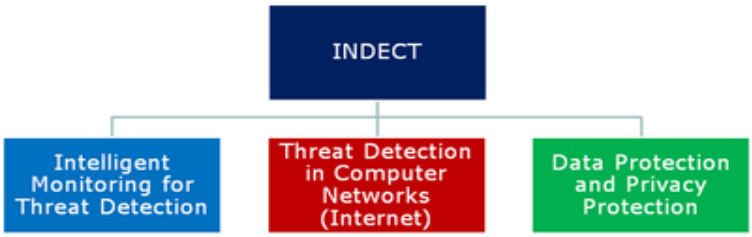


Figura 3. Principales tareas de investigación en INDECT [1].

La finalidad de dichas herramientas es facilitar la obtención y análisis de información, por ejemplo mediante cámaras de seguridad, automatizando estas tareas lo más posible para que finalmente un ser humano capacitado (policía, servicios de seguridad,…) decida si es necesario intervenir. Esto permite menos intervención humana en el proceso, lo que disminuye la subjetividad, los errores humanos y la posibilidad de que la información sea usada con fines ilegítimos o se produzcan violaciones de los derechos individuales. Estas herramientas quedarán englobadas en una plataforma para el registro e intercambio de datos asociados con el reconocimiento de amenazas, la adquisición de contenido multimedia, el procesado inteligente de información relacionada con la detección automática de amenazas y el reconocimiento de comportamientos criminales o violentos. La definición de las situaciones y los parámetros utilizados para realizar la automatización está siendo proporcionada por las LEAs.

Debido a que ha habido cierta controversia respecto a los temas éticos del proyecto, ya que éste implica el tratamiento de información sensible, desde vídeos de seguridad a contenido multimedia en redes P2P (*peer to peer*) o datos almacenados en servidores web, consideramos importante hacer un inciso y mencionar que se cumplen estrictamente las regulaciones de la EU sobre privacidad y protección de datos entre otros. El proyecto incluye un comité ético cuya finalidad es asegurar el cumplimiento de normas tanto nacionales como internacionales como son:

- La carta de los derechos fundamentales de la Unión Europea (*EU Charter of Fundamental Rights*) firmada el 7 de diciembre de 2000 en Niza.
- Convención Europea de Derechos Humanos (*European Convention of Human Rights*) suscrita el 4 de noviembre de 1950 en Roma.
- Ley inglesa de protección de datos del 16 de julio de 1998 (*Data Protection Act 1998*).
- Ley inglesa de los derechos humanos del 9 de noviembre de 1998 (*Human Rights Act 1998*).

- Ley inglesa sobre los poderes inquisitoriales del 28 de Julio del 2000 (*Regulation of Investigatory Powers Act*).

El proyecto INDECT se divide en varios paquetes de trabajo, en varios de los cuales está presente la Universidad Carlos III de Madrid. El desarrollo sobre el que trata este proyecto fin de carrera se incluye concretamente en el tercero de ellos, denominado WP3. Como ya se ha comentado, la plataforma ILIP se divide en dos sistemas: una estación de captura que se encarga de obtener el tráfico y mandárselo a otra estación decodificadora cuya función es analizarlo y clasificarlo. Todo el proceso se desarrolla asegurando que se cumplan los requisitos de seguridad necesarios, prestando especial énfasis en asegurar el cumplimiento de la normativa en cuanto a interceptación legal de comunicaciones, a la vez que se aportan soluciones innovadoras. Anteriormente ya se habían desarrollado varias aplicaciones software independientes con el objeto de dotar de mayor funcionalidad a la plataforma ILIP y el objetivo de este proyecto es integrarlas para crear la estación decodificadora. Tanto la estación de captura como el diseño del mecanismo de seguridad se encuentran también en proceso de diseño y desarrollo por personal de la Universidad Carlos III de Madrid, pero no se detallarán puesto que quedan fuera del ámbito de este proyecto.

La estación decodificadora de la plataforma ILIP se basa en la herramienta de análisis forense *Xplico* [4], que decodifica el tráfico de red capturado y proporciona información sobre el contenido intercambiado. Algunas de las aplicaciones que se han desarrollado permiten extender su funcionalidad mediante el procesado y etiquetado de dichos contenidos:

- Herramienta LINK [5] para el análisis de grupos de crimen organizado, que también ha sido desarrollada dentro el grupo de trabajo WP3, pero por la Universidad de AGH.
- Módulo de distribución y preclasificación de contenidos [5] que permite agilizar el procesado de los contenidos al distribuirlos entre las distintas aplicaciones encargadas de realizar el etiquetado o preclasificación. Proyecto fin de carrera desarrollado por Rubén Mena Escribano.
- Módulo de transcripción VoIP-texto [7] que obtiene transcripciones textuales a partir de llamadas de voz sobre IP. Proyecto fin de carrera desarrollado por Jesús Vallinot Sánchez.
- Módulo de identificación de contenidos basado en hashes difusos [8] que realiza la preclasificación de contenidos combinando la técnica de *fuzzy hashing* y listas negras. Proyecto fin de carrera desarrollado por Roberto Sánchez.
- Herramienta de procesado de imágenes INACT (*INDECT Advance Image Catalog Tool*) [9], desarrollada en el grupo de trabajo WP5.

De momento sólo las tres primeras herramientas han sido integradas en el sistema por lo que las dos últimas pertenecen a trabajos futuros. Partiendo de la herramienta *Xplico*, el primer paso será incrementar su seguridad para integrar a continuación las aplicaciones listadas una a una, manteniéndolas como módulos separados para facilitar futuras actualizaciones y ampliaciones de funcionalidad. Al final obtendremos un sistema seguro y modular de análisis de contenidos que conformará la estación decodificadora.

1.3 Esquema de la memoria

Esta memoria se ha estructurado de la siguiente manera:

- Capítulo 2: Estado del arte. Consiste en un resumen de los estándares relacionados con la interceptación legal de comunicaciones que se han considerado importantes para el proyecto y la presentación de la aplicación software sobre la que se basa la estación decodificadora, *Xplico*.
- Capítulo 3: Diseño e Implementación. En sus diferentes apartados se explican los diferentes módulos y el proceso relacionado para su integración en la estación decodificadora.
- Capítulo 4: Evaluación y Presupuesto.
- Capítulo 5: Conclusiones y trabajos futuros.
- Anexos: para no entrar en excesivo detalle a lo largo de la memoria, se ha decidido incluir cierta información técnica en forma de anexos, de forma que pueda consultarse si se desea:
    - Anexo I: CakePHP
    - Anexo II: Certificados digitales X.509. Instalación de un servidor ILIP seguro mediante certificados digitales X.509 y OpenSSL.
    - Anexo III: LDAP y LDIF
    - Anexo IV: Ejemplos de *plugins*.
    - Anexo V: Instalación de un servidor de *plugins* remotos mediante Apache Tomcat y Axis2. Paquete Java para desarrolladores de *plugins*.

Capitulo 2: **Estado del arte**

En este apartado se estudiarán dos puntos muy importantes para este proyecto. Primero se mencionará brevemente cómo fueron surgiendo y avanzando los estándares del ETSI sobre la interceptación legal de comunicaciones y se tratarán los más actuales de entre los que influyen en nuestro trabajo. Posteriormente se estudiará la herramienta forense Xplico, que forma los cimientos de la estación decodificadora de ILIP.

2.1 Estándares de Interceptación Legal de Comunicaciones

Dentro del Instituto Europeo de Estándares de Telecomunicaciones (ETSI), el grupo de trabajo en técnicas de seguridad (*Security Techniques Advisory Group* o STAG) produjo en Diciembre de 1996 el informe técnico ETR 331 [10] (*ETSI Technical Report*) debido a la necesidad de estandarización en el área de la interceptación legal de las comunicaciones (*Lawful Interception* o LI). Anteriormente a este estándar los documentos acerca de la LI básicamente proporcionaban información acerca de la seguridad en las telecomunicaciones y los aspectos regulatorios de la misma, como podemos comprobar viendo cuáles son las referencias de este estándar: la resolución del Consejo de la Unión Europea sobre la interceptación legal de las telecomunicaciones del 17 de enero de 1995 [11] y el ETR 330 [12] producido en noviembre de 1996, que básicamente es un documento informativo para ser usado como referencia. El objetivo del ETR 331 era definir e iniciar el camino hacia un diseño técnico de una interfaz entre las operadoras y las agencias de seguridad (LEAs) en un estándar europeo de telecomunicaciones (*European Telecommunication Standard* o ETS). Este proceso está compuesto por los siguientes pasos:

- Paso 1: La definición de los requisitos de usuario, esto es, las LEAs. Este paso se realizó ya dentro de este mismo ETR que lo define, para lo cual se requirió cooperación con las LEAs.
- Paso 2: La definición de los requisitos para los operadores de red, también con cooperación de las mismas. A su vez supone la etapa 1 de la descripción de la interfaz de entrega HI.
- Paso 3: Cubre las etapas 2 y 3 de la descripción de la interfaz de entrega HI, obteniendo modelos concretos para servicios y productos específicos.

Prácticamente año y medio después, en mayo de 1998, se publicó el estándar del ETSI ES 201 158 [13], cumpliendo el paso 2 al definir los requisitos desde el punto de vista de los operadores de red y en julio de 1999 se publicó el estándar ES 201 671 [14] finalizando los pasos al describir por primera vez una interfaz de entrega genérica entre los operadores y las LEAs, especificando un flujo genérico de información y los procedimientos y elementos de información necesarios en cualquier servicio o red de comunicaciones tanto de aquel momento como futuras y los detalles específicos para algunos de los servicios de aquel momento. A lo largo de los años, estos tres documentos han ido mejorando y siendo actualizados según iban estandarizándose nuevas tecnologías. A fecha de hoy, y en lo que se refiere a este proyecto, las versiones más recientes de los estándares analizados son:

- TS 101 331, V1.3.1, de octubre de 2009 [15]: Requisitos de las LEAs.
- ES 201 158, V1.2.1, de abril de 2002 [16]: Requisitos de los operadores de red.

- TS 101 671, V3.10.1, de junio de 2012 [17]: Descripción de la interfaz de entrega.

Por supuesto, han ido apareciendo otros documentos, tales como el TR 101 943, cuya primera versión apareció en julio del 2001 y la más actual data de noviembre del 2006 [18]. Este documento pretende ser una guía que cubre algunos aspectos prácticos de la implementación de sistemas de LI.

Los documentos hasta ahora mencionados tratan el tema de la LI desde una perspectiva general, independientemente de la tecnología particular (aunque incluyen algunos anexos con información específica). Este proyecto está basado en la LI en redes IP, por lo que los estándares particulares para esta tecnología son:

- TR 101 944, V1.1.2, de diciembre de 2001 [19]: Trata sobre problemas técnicos existentes en la LI en redes IP, básicamente el problema de identificar los paquetes IP correspondientes al sujeto bajo investigación.
- TR 102 528, V1.1.1, de octubre de 2006 [20]: Orientada a los operadores de red, proporciona una arquitectura general de referencia para realizar la interceptación en redes IP.

Además, y aunque no se entrará en más detalle sobre ellos, cabe mencionar el estándar TS 102 232, compuesto por 7 documentos:

1) TS 102 232-1, V3.1.1, de junio de 2012 [21]: Describe cómo entregar información a través de redes IP.
2) TS 102 232-2, V3.1.1, de febrero de 2012 [22]: Describe la información necesaria en la entrega de tráfico de mensajería, incluyendo en este contexto *email*, *webmail* o SMS, por ejemplo.
3) TS 102 232-3, V3.2.1, de junio de 2012 [23]: Se centra en la LI de datos IP en relación con el uso de servicios de acceso a internet (*Internet Access Services* o IAS).
4) TS 102 232-4, V3.1.1, de febrero de 2012 [24]: Especifica la LI para un proveedor de acceso que tiene acceso a la información de sesión de la capa 2 pero no superiores.
5) TS 102 232-5, V3.2.1, de junio de 2012 [25]: Especifica la LI de servicios multimedia basados en los protocolos SIP (*Session Initiation Protocol*), RTP (*Real Time Transport Protocol*) o MSRP (*Message Session Relay Protocol*).
6) TS 102 232-6, V3.1.1, dejunio de 2012 [26]: Describe la información necesaria en la entrega de información interceptada de servicios PSTN/ISDN (*Public Switched Telephone Network/Integrated Services Digital Network*) o de servicios emulados usando técnicas basadas en paquetes.
7) TS 102 232-7, V3.1.1, de junio de 2012 [27]: Especifica una aproximación a la entrega de información interceptada de servicios móviles.

A continuación se presenta un resumen de la información que se ha considerado relevante para el proyecto, indicando los estándares de los que se ha obtenido.

2.1.1 Requisitos de las agencias de seguridad

El documento TS 101 331 proporciona una serie de requisitos para la LI desde el punto de vista de las LEAs. Estos requisitos están sujetos a leyes nacionales y tratados internacionales por lo que deben ser interpretados en su contexto particular e implementados si la ley así lo requiere. Los siguientes puntos constituyen un resumen de los que se han considerado más importantes en el contexto del proyecto:

- La obligación de interceptar tráfico por parte del operador de red (*network operator* o NWO), proveedor de acceso (*access provider* o AP) o proveedor de servicios (*service provider* o SvP) depende de las leyes nacionales.
- Una autorización legal especifica los requisitos particulares de la interceptación a un determinado sujeto.
- Los resultados de la interceptación se relacionan con la autorización mediante una identificación única.
- El proveedor cooperará inmediatamente al recibir una autorización legal.
- Todo el contenido de la comunicación asociado a un sujeto bajo vigilancia podrá ser interceptado pero sólo lo especificado en la autorización legal será enviado a las instalaciones de la LEA, denominadas LEMF (*Law Enforcement Monitoring Facilities*).
- El envío de información será fiable, y lo más pronto posible.
- El proveedor no monitorizará o grabará permanentemente los resultados de la interceptación.
- Los resultados de la interceptación están formados por el propio contenido de la comunicación (*contents of communication* o CC) e información relacionada *(intercept related information* o IRI), como pueden ser las identidades de los sujetos que han intentado contactar (con o sin éxito) con el sujeto bajo investigación o viceversa, otras identidades objetivo asociadas al sujeto bajo investigación, información del servicio o estado, localización, etc.
- La información enviada a la LEA estará libre de cualquier codificación o cifrado aplicado por el proveedor, o éste proporcionará las claves necesarias para descifrarlo.
- Cada sujeto bajo vigilancia estará asociado con una instancia de la interfaz de entrega (*handover interface* o HI) que comunica el proveedor y la LEA.
- La HI estará configurada de tal manera que la transmisión de los resultados pueda hacerse mediante protocolos o codificación estándar.
- Si existen varias LEAs investigando a un mismo sujeto, el proveedor debe asegurar la confidencialidad de sus identidades e investigaciones.

2.1.2 Requisitos de las redes

El documento ES 201 158 proporciona una serie de requisitos para la LI desde el punto de vista de los operadores de red (NWO), proveedores de acceso (AP) y de servicio (SvP). Dado que la forma concreta de realizar el acto de la interceptación depende del operador, este estándar trata principalmente del proceso a llevar a cabo para iniciar una LI, y de la posterior entrega y formato de los resultados obtenidos. Toda la comunicación se realiza a través de la interfaz de entrega (HI) cuyos detalles se posponen al apartado siguiente.

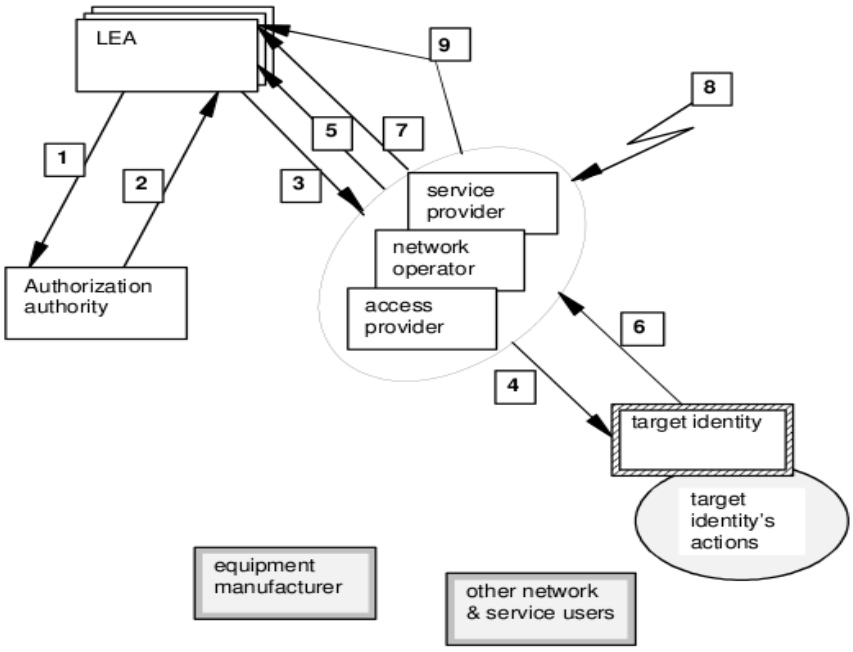


Figura 4. Modelo del proceso de interceptación legal de comunicaciones [16].

A continuación se especifica el proceso completo de interceptación legal de comunicaciones mostrado en la Figura 4, indicando los pasos que hay que realizar desde que una LEA solicita la interceptación legal de un sujeto hasta que recibe los resultados de dicha interceptación:

1) La LEA pide una autorización legal a una autoridad de autorización.
2) La autoridad de autorización concede la autorización legal al LEA.
3) La LEA envía la autorización legal al NWO/AP/SvP, que le permite determinar las identidades internas del sujeto o identidades objetivo.
4) El NWO/AP/SvP empieza la interceptación de las identidades objetivo.
5) El NWO/AP/SvP informa a la LEA de que la autorización se recibió y se ha actuado en consecuencia. También se puede enviar información adicional acerca de las identidades objetivo.
6) EL NWO/AP/SvP obtiene el resultado de la interceptación de dichas identidades.
7) EL NWO/AP/SvP envía el resultado de la interceptación al LEMF de la LEA.
8) Al finalizar el periodo de interceptación o bajo petición de la LEA, se finaliza la interceptación.
9) EL NWO/AP/SvP anuncia a la LEA el fin de la interceptación.

En cuanto a las autorizaciones legales, la ley nacional describe las condiciones y requisitos bajo los que se permite la interceptación. La LEA debe pedir a la autoridad correspondiente como mínimo una autorización judicial por sujeto y NWO/AP/SvP que necesite que ejecute la interceptación. En cada orden se especifican: el sujeto bajo investigación, la duración de la interceptación, el tipo de información que se debe entregar al LEA y las direcciones del LEMF donde entregar la información al mismo, entre otros. Sin embargo, no hay una forma única de identificar el sujeto. En países en los que exista un documento legal de identidad, éste sirve como identificador unívoco dentro del país pero puede que éste no sea válido fuera del mismo. En ese caso, se podría utilizar la dirección postal o cualquier otro dato que permita al NWO/AP/SvP saber exactamente de quién se trata para posteriormente utilizar sus identificadores internos.

Los resultados de la interceptación se dividen en dos tipos:

- El contenido de la comunicación (*content communication* o CC) intercambiado entre el sujeto bajo investigación y otras entidades.
- La información relacionada con la interceptación (*intercept related information* o IRI) referente al servicio utilizado: identidad del sujeto, información asociada con la comunicación, el propio servicio o la localización del sujeto.

Debido a que la información de cada tipo a entregar al LEA viene especificada en la autorización legal, es posible que varios LEAs investiguen a la vez a un mismo sujeto, de forma que las leyes o restricciones respecto al IRI y CC que apliquen a cada uno sean diferentes.

El bloque de fabricantes de equipos (*equipment manufacturer* en la Figura 4) representa el equipamiento usado en la infraestructura LI de los NWO/AP/SvP. Tanto éste como el bloque de otros usuarios de red y servicio (*other network and service users* en la Figura 4) se presentan aislados, representando que ni los fabricantes ni ningún otro usuario debe poder detectar que una interceptación ha sido activada o desactivada, o está teniendo lugar, ni tampoco ser sus comunicaciones interceptadas a no ser que su comunicación sea con el sujeto bajo vigilancia.

Ya que un SvP hace uso de otros APs y NWOs y un sujeto puede hacer a su vez uso de varios SvPs, la cooperación entre ellos puede ser necesaria. También es posible que la provisión de un servicio implique cruzar fronteras, en cuyo caso se deben aplicar las leyes nacionales y tratados internacionales correspondientes.

En cuanto a la seguridad de la plataforma de LI, destacan los siguientes puntos:

1) Debe haber controles tanto físicos como lógicos.
2) Los datos necesarios para la autorización y acceso a las funciones de interceptación deben ser almacenados y transmitidos de forma segura.
3) Ambos extremos deben ser capaces de autenticarse mutuamente. Si la autenticación falla la conexión debe rechazarse y se debe generar un informe.
4) El número de elementos en los que se almacena, administra o procesa información relacionada con la interceptación debe ser el mínimo posible.
5) La LEA no debe tener acceso a ningún elemento de red.
6) El acceso a la función de interceptación sólo debe permitirse desde localizaciones específicas.

2.1.3 Interfaz de entrega

Los documentos presentan una estructura genérica de la interfaz de entrega HI que comunica los NWOs/APs/SvPs con las LEAs, con la intención de que pueda ser aplicada a cualquier sistema de telecomunicaciones, aunque es susceptible de ser revisada o mejorada. Mientras que el documento ES 201 158 proporciona una descripción básica, el documento TS 101 671 realiza una especificación detallada de la misma y el TR 101 943 pretende ser una guía que cubre algunos aspectos prácticos de la implementación de sistemas de LI. En este apartado se presenta un resumen de los tres documentos, empezando por una descripción más general y entrando poco a poco en más detalle.

El HI debe ser configurable para adaptarse a requisitos y leyes nacionales así como a las leyes aplicables a una LEA particular. Por seguridad, la LEMF y el NWO/AP/SvP deberían autenticarse antes de transmitir información y dicha información debe transmitirse de manera cifrada. Dado que los resultados de la interceptación se basan en identidades técnicas internas al proveedor, la información deberá estar etiquetada de manera que se pueda relacionar con la autorización legal correspondiente a la LI y con el sujeto bajo vigilancia.

La HI se divide en tres puertos, según el tipo de información que se transmita por ellos:

- El primer puerto, denominado HI1, es la interfaz administrativa, por la que se envían las autorizaciones judiciales.
- El segundo puerto, denominado HI2, transporta la información relacionada con la interceptación (*intercept related information* o IRI).
- El tercer puerto, denominado HI3, transporta el contenido de la comunicación (*content of communication* o CC*).*

Estas interfaces pueden ser manuales, basadas por ejemplo en la entrega de documentos, o electrónicas. Uno de los detalles más importantes es que HI1 esté totalmente separada de HI2 y HI3 para evitar que una LEA pueda establecer o modificar la interceptación directamente. Sin embargo, HI2 y HI3 pueden combinarse perfectamente en una única interfaz. En cualquier caso, IRI y CC deben estar correladas para que la LEA pueda relacionarlas. Esto puede hacerse por ejemplo mediante marcas de tiempo, identificadores únicos, la dirección del LEMF o el uso de un canal físico particular.

HI1 transporta cualquier tipo de información administrativa entre el NWO/AP/SvP y la LEA, como por ejemplo la autorización legal de la LEA para una determinada interceptación, la petición de la LEA de iniciarla, el acuse de recibo del operador o la petición de parar la interceptación antes del tiempo indicado en la autorización. También transporta informes de estado y alarmas tales como que la identidad haya sido eliminada del servicio o haya cambiado dentro de la red, que alguno o todos los números de suscriptor del sujeto han cambiado, o cualquier tipo de fallo general o de transmisión a la LEA (ocupado, sin respuesta...). Para poder activar la interceptación, la LEA debe proporcionar como mínimo: la autorización legal especificando la identidad del sujeto bajo vigilancia, el inicio y fin de la interceptación, el tipo de IRI y CC que debe recibir y las direcciones destino de la LEMF donde enviarlos, un identificador de interceptación

legal (*Lawful Interception Identifier* o LIID) acordado entre ambas partes del que se hablará más adelante y algún tipo de contacto técnico.

HI2 transporta la IRI desde el operador a la LEMF. La IRI incluye información sobre identificadores del sujeto, información de señalización para establecer y controlar el progreso del servicio, o información de localización. Cuando se envíe IRI relacionada con la comunicación, sólo será información que forme parte de los procesos estándar de señalización de la red, no se exigirá información que requiera procedimientos extra. En general debe transmitirse según se genera aunque en algunos casos se puede admitir cierto almacenamiento intermedio y la agregación de distintos IRIs que vayan destinados a la misma LEMF. El formato deberá basarse en estándares si es posible.

HI3 transporta el CC desde el operador a la LEMF. Debe ser una copia en claro de la información enviada durante la comunicación del sujeto, por lo que si el operador utiliza cifrado, debe decodificarla previamente o proveer a la LEA con las claves necesarias.

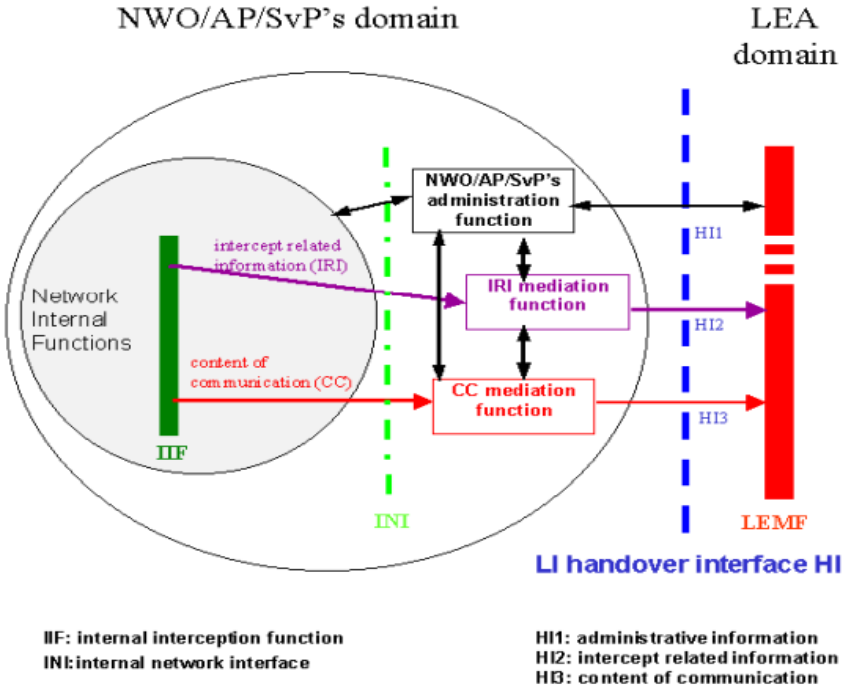


Figura 5. Diagrama de bloques funcional de la interfaz de entrega [17].

En la Figura 5 se puede apreciar la estructura interna de la HI en el dominio del NWO/AP/SvP. El círculo externo representa dicho dominio; internamente aparece un círculo más pequeño que contiene las funciones internas de la red (conmutación, reenvío, enrutamiento,...) dónde se genera la IRI y el CC. Esta información es capturada por las denominadas funciones internas de interceptación (*internal interception función* o IIF) que también son propias de la red. La interfaz interna de red (*internal network interface* o INI) permite el envío de esta información a otras funciones que no pertenecen a la red, y que se encargan de realizar la comunicación con la LEA a través de los tres puertos de la HI: son las funciones de administración y de mediación de la IRI y del CC. La función administrativa (*administration function* o AF) recibe las peticiones de la LEA y transmite informes de estado y alarmas. La función de mediación (*mediation function* o MF) transforma la información (IRI o CC) del formato interno al formato requerido por la LEA, pudiendo darse el caso de que no sea necesaria dicha transformación.

Para poder identificar de forma única el sujeto que está siendo interceptado y poder relacionar los datos en los tres puertos de la HI es necesario utilizar una serie de identificadores:

1) Identificador de interceptación legal (*lawful interception identifier* o LIID)

   Acordado entre la LEA y el NWO/AP/SvP. Se utiliza para identificar indirectamente al sujeto bajo vigilancia, ya sea un identificador para cada una de las identidades técnicas del sujeto y/o uno único para todas. Consiste en una cadena de caracteres alfanuméricos, como puede ser el número de referencia de la autorización legal, seguido de la fecha en que se concedió.

2) Identificador de la comunicación (*communication identifier* o CID)

   Identifica la actividad del sujeto y es generado por el elemento de red que realizó la interceptación. Está formado a su vez por uno o dos identificadores: un identificador de red y opcionalmente un número de identidad de comunicación.

3) Identificador de red (*network identifier* o NID)

   Debe ser único internacionalmente. Está formado por uno o dos identificadores: el identificador de NWO/AP/SvP y opcionalmente el identificador del elemento de red (*network element identifier* o NEID) que está realizando la tarea, como podría ser su dirección IP.

4) Número de identidad de comunicación (*communication identity number* o CIN)

   Identifica una sesión dentro del elemento de red. Por ejemplo, si un sujeto tiene varias sesiones en el mismo elemento de red, deben tener un CIN diferente; pero si varios sujetos se comunican entre ellos dentro de la misma sesión, el mismo CIN debe ser asignado a ambos. Es obligatorio en los mensajes de la IRI en el caso de que la comunicación sea orientada a conexión.

5) Número de correlación (*correlation number*)

   Es específico para redes de paquetes conmutados. Es único por protocolo y se utiliza para correlar el CC con la IRI o varios IRI entre sí.

En cuanto a seguridad, los canales que transmitan información deben cumplir las siguientes propiedades:

- Confidencialidad (no es posible interpretar los datos), por ejemplo mediante cifrado.
- Integridad (cualquier alteración de los datos es inmediatamente detectada), por ejemplo mediante algoritmos de *hashing*.
- Autenticación (ambos extremos de la comunicación verifican la identidad del otro), por ejemplo mediante técnicas criptográficas.
- Disponibilidad (ambas partes acuerdan tiempos de funcionamiento y parada).

Anteriormente los servicios y la tecnología estaban muy ligados entre sí. Sin embargo la tendencia es que cada vez sean más independientes, de manera que varios servicios puedan ser proporcionados sobre la misma tecnología. Se definen tres capas en un sistema de comunicaciones: de servicio, de control y de conectividad. La capa de servicio proporciona el servicio específico en sí y se encarga de la autenticación y

autorización, envía una petición a la capa de control que se encarga de la parte técnica del servicio, que a su vez envía peticiones a la capa de conectividad para establecer conexiones. La Figura 6 muestra la relación entre dichas capas con las funciones de LI.

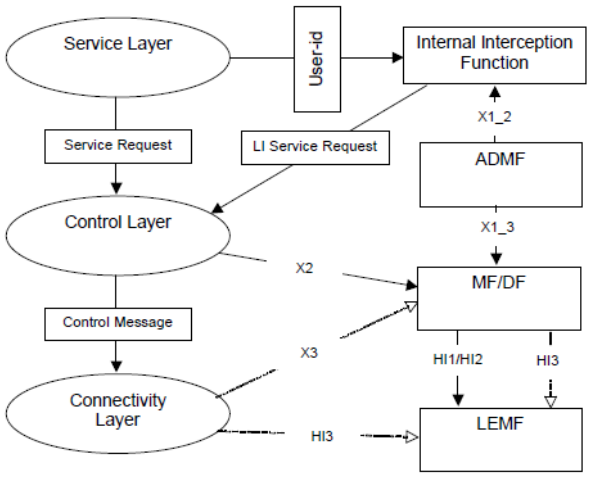


Figura 6. Relación entre las capas de un sistema de comunicaciones y los elementos que realizan la interceptación [18].

La capa de servicio permite identificar el usuario, la de control proporciona la IRI y la capa de conectividad proporciona el CC. La capa de servicio compara la identidad del usuario que se conecta con una lista de identidades objetivo y si coincide con alguna, envía una petición de interceptación a la capa de control. Esta capa empieza a enviar mensajes de IRI a la MF e indica a la capa de conectividad que capture y envíe la información de CC a la LEMF. Otra forma de verlo es mediante la torre de protocolos TCP/IP. La Figura 7 muestra la interacción entre las capas en la interceptación de CC.

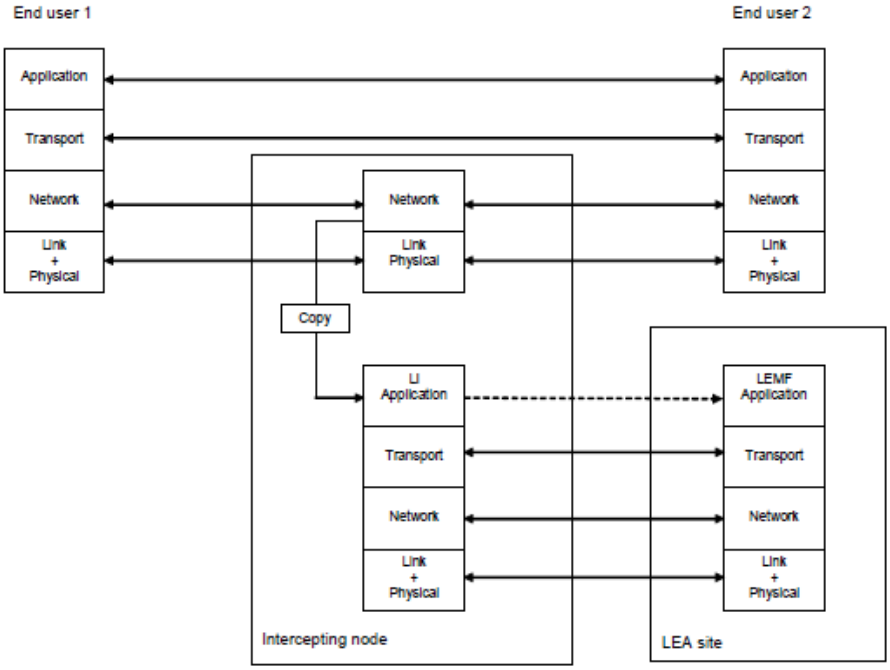


Figura 7. Interacción de la torre de protocolos para la interceptación del contenido de la comunicación [18].

2.1.4 Arquitectura de Interceptación Legal en redes IP

El documento TR 102 528 describe una arquitectura de referencia para realizar la LI en redes IP; esta se muestra en la Figura 8. Aunque en dicho documento se realiza una explicación individual de los elementos funcionales que se observan en dicha figura, se ha preferido exponerlos según los pasos del proceso de la LI, de forma que quede clara la manera en que los bloques interactúan entre sí. En cualquier comunicación entre bloques será necesario enviar, además de los datos que se indiquen a lo largo de la explicación, el LIID, las credenciales de seguridad y otros parámetros, como por ejemplo, de transporte.

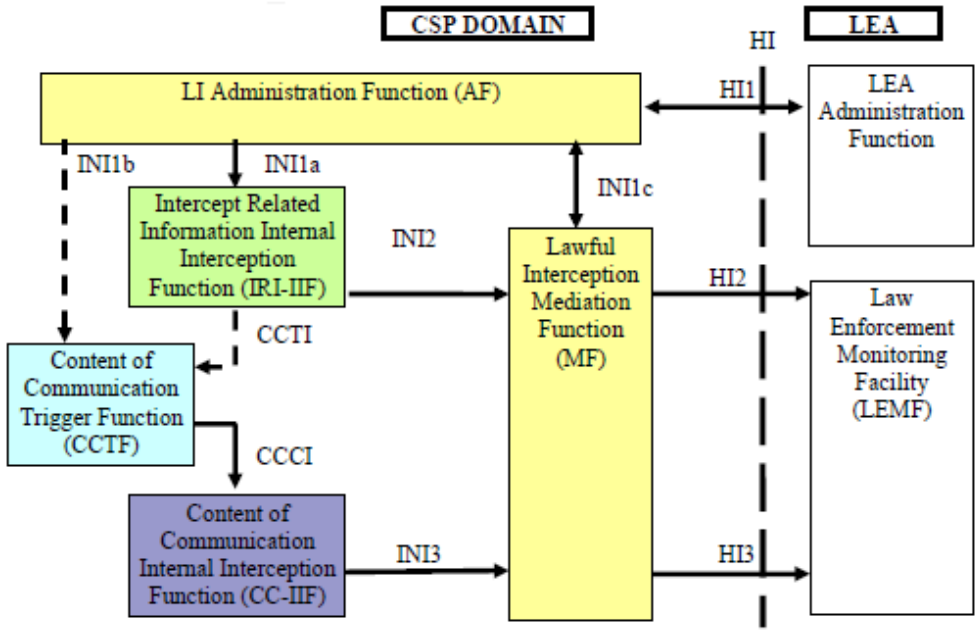


Figura 8. Modelo funcional de referencia para la interceptación legal de comunicaciones [20].

Cada elemento lógico representa una función dentro de la arquitectura y por lo tanto sólo aparece una vez. Sin embargo, un mismo elemento físico dentro del operador puede realizar varias funciones a la vez o si la red es grande, como suele ser el caso, puede haber varios elementos físicos desarrollando la misma función pero en distintas localizaciones. En este último caso será necesario enviar también información que permita identificar al elemento de red correspondiente.

La LEA envía a la operadora la autorización legal y cualquier otro dato necesario, como la dirección destino de la LEMF donde enviar los resultados de la interceptación. Este contacto se realiza entre las funciones administrativas (AF) de ambas a través del puerto HI1. La operadora comprueba la información recibida, obtiene las identidades objetivos correspondientes y envía confirmación a la LEA adjuntando cualquier información que sea necesaria, como dichas identidades. Una vez la legalidad de la interceptación ha sido confirmada la AF se pone en contacto con tres elementos mediante la interfaz interna de red 1 (*internal network interface 1* o INI1):

- A través de INI1a se provisiona a la función interna de interceptación de IRI (*IRI internal interception function* o IRI-IIF) con la información estática necesaria sobre el sujeto, para que ésta pueda generar la IRI asociada con sesiones, llamadas, conexiones y cualquier otra información relacionada.

- A través de INI1b se provisiona a la función disparadora de CC (*CC trigger function* o CCTF) con la información estática disponible acerca del sujeto y cualquier otra información necesaria.
- A través de INI1c se provisiona a la función de mediación (*mediation function* o MF) con la dirección de la LEMF a la que enviar los resultados de la interceptación, la información necesaria para que pueda discernir el IRI y el CC del sujeto entre todos los que reciba y datos sobre el formato que el LEA requiere si alguna transformación es necesaria.

La IRI obtenida se envía a la función de mediación (*mediation function* o MF) a través de la interfaz INI2, conjuntamente con los identificadores necesarios para que éste pueda identificar y correlar toda la información relacionada, y se utiliza para obtener información dinámica (como direcciones IP o puertos) que se proporciona al CCTF a través de la interfaz disparadora de CC (*CC trigger interface* o CCTI). Una vez que dispone de la información suficiente, el CCTF determina la localización del elemento que realizara la LI y se pone en contacto con su correspondiente función interna de interceptación (*CC internal interception function* o CC-IIF) a través de la interfaz de control del CC (*CC control interface* o CCCI), para enviarle la información necesaria tanto para filtrar el sujeto como para controlar el proceso. La CC-IIF empieza entonces a realizar la interceptación de las comunicaciones del sujeto. Una vez se obtiene el CC, se envía a la MF a través de la interfaz INI3, de nuevo con los identificadores necesarios.

Por último la MF correla y formatea la IRI y el CC para enviarlos a las correspondientes LEMF a través de las interfaces HI2 y HI3, respectivamente.

Aunque no se ha entrado en detalles acerca del tipo y formato de los mensajes de la IRI y el CC, a continuación se adjuntan y explican dos figuras que permiten ver el intercambio de mensajes que se produce entre elementos durante la interceptación. La Figura 9 es un ejemplo de interceptación del envío de mensajes multimedia entre el sujeto y otro usuario. Los primeros mensajes intercambiados entre ambos son mensajes SIP de control, para definir entre otros el puerto sobre el que se realizará la comunicación. Una vez se han acordado los detalles, comienza el flujo de paquetes RTP que llevan los datos intercambiados.

En este caso lo más sencillo es que el elemento IRI-IIF se corresponda con el elemento de la red que realiza el control de las llamadas, de manera que se detecte cada vez que el sujeto intenta llamar o iniciar sesión, lo que permite obtener los datos necesarios para interceptar la comunicación. La AF y la MF se representan como un mismo elemento, que además realiza la función de la CCTF. En este caso, en lugar de realizar directamente peticiones de interceptación tanto a la IRI-IIF como a la CCTF, se realiza una petición inicial al primero, ya que es necesario obtener algunos datos dinámicos para poder interceptar el CC. Una vez obtenidos estos datos puede realizarse la petición directamente a la CC-IIF.

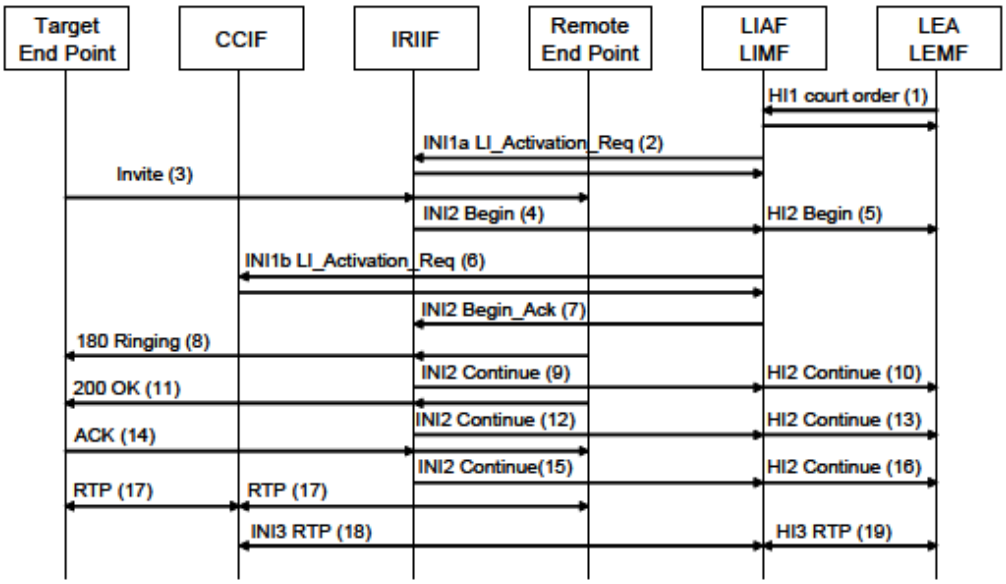


Figura 9. Flujo de mensajes en la interceptación legal de una sesión multimedia [20].

Los pasos del proceso de LI son los siguientes:

1) La LEA entrega la orden de autorización a través de HI1.
2) La AF envía a la IRI-IIF una petición de activación de LI a través de INI1a.
3) El sujeto envía una invitación SIP a otro usuario. Este mensaje es interceptado por el IRI-IIF y analizado para obtener la dirección IP del sujeto y el puerto UDP que se va a utilizar para la comunicación.
4) La IRI-IIF envía un mensaje con datos de IRI a la MF a través de INI2.
5) La MF reenvía el mensaje a la LEMF a través de HI2.
6) La AF envía al CC-IIF una petición de activación de LI a través de INI1b.
7) La MF confirma que ha recibido el mensaje de la IRI a través de INI2.
8) El sujeto iniciar una llamada mediante un mensaje SIP y es interceptado por la IRI-IIF.
9) La IRI-IIF envía otro mensaje de IRI a la MF a través de INI2.
10) LA MF lo reenvía a la LEMF a través de HI2.
11) El otro usuario responde y el mensaje SIP es interceptado también.
12) La IRI-IIF envía otro mensaje de IRI a la MF a través de INI2.
13) LA MF lo reenvía a la LEMF a través de HI2.
14) El sujeto envía un mensaje SIP que es interceptado.
15) La IRI-IIF envía otro mensaje de IRI a la MF a través de INI2.
16) LA MF lo reenvía a la LEMF a través de HI2.
17) La comunicación entre el sujeto y el otro usuario se realiza mediante paquetes UDP/RTP que son interceptados por la CC-IIF.
18) La CC-IIF envía mensajes de CC con el contenido RTP al MF a través de INI3.
19) LA MF reenvía los mensajes CC a la LEMF.

La Figura 10 es un segundo ejemplo para el caso de un sujeto que se conecta a un servidor para autenticarse y posteriormente acceder a un servicio. Dicho servidor hace las funciones de IRI-IIF. Igual que en el caso anterior es necesario obtener primero alguna información dinámica, pero en este caso se realizara el proceso a través del elemento CCTF. Los pasos son los siguientes:

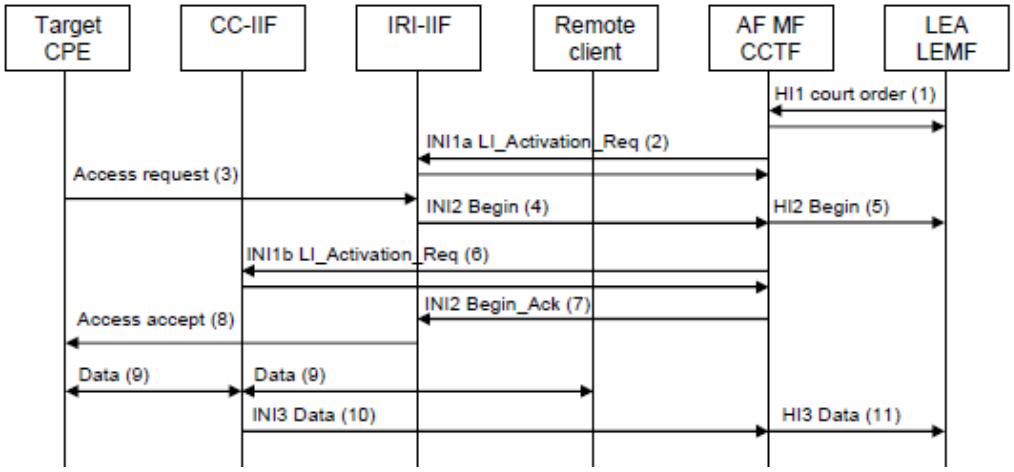


Figura 10. Flujo de mensajes en la interceptación legal de una conexión de datos básica [20].

1) La LEA entrega la orden de autorización a través de HI1.
2) La AF envía a la IRI-IIF una petición de activación de LI a través de INI1a.
3) El sujeto envía una petición de acceso y sus datos de autenticación al servidor. La petición de acceso es interceptada por el IRI-IIF.
4) La IRI-IIF envía un mensaje de IRI a la MF a través de INI2.
5) La MF lo reenvía a la LEMF a través de HI2.
6) La AF envía al CC-IIF una petición de activación de LI a través de INI1b.
7) La MF confirma que ha recibido el mensaje de la IRI a través de INI2.
8) Se confirma la petición de acceso del usuario.
9) Comienza la sesión de datos del usuario.
10) La CC-IIF envía mensajes de CC con el contenido de dicho flujo al MF a través de INI3.
11) La MF reenvía los mensajes CC a la LEMF.

2.2 Herramientas forenses de análisis de red: *Xplico*

Para desarrollar la estación decodificadora de ILIP se decidió partir de una herramienta de análisis forense de tráfico de red y modificarla añadiendo la funcionalidad necesaria. Algunas conocidas herramientas de análisis de tráfico como *Wireshark* o *TCPdump* sólo proporcionan información sobre los protocolos de red por lo que no se ajustaban a las necesidades del sistema. Era necesario conseguir extraer los datos de nivel de aplicación contenidos en una captura de tráfico de internet, que es precisamente lo que hace *Xplico* [4].

Xplico ha sido desarrollada por Gianluca Costa y Andrea De Franceschi. Actualmente la versión estable es la 1.0.1. Esta herramienta está siendo desarrollada como código abierto bajo la licencia *GNU General Public License* (GPL) con una implementación totalmente modular que nos permite modificar la aplicación para poder realizar la integración con el software desarrollado específicamente para ello. En este apartado se hablará sobre la funcionalidad y la interfaz gráfica de la herramienta y más adelante, cuando sea necesario para entender las decisiones tomadas, se entrará en detalle acerca del funcionamiento interno de *Xplico*. Está desarrollada con elementos en diferentes lenguajes de programación como son C, Python, PHP y Javascript. Para empezar a manejar *Xplico*, lo primero que hay que hacer es crear un caso y dentro de él al menos una sesión, donde cada sesión contiene flujos de tráfico relacionados entre sí, como por ejemplo, que hayan sido capturados el mismo día o en el mismo operador. Una vez que el usuario introduce una captura de tráfico en una sesión entonces la herramienta procede a realizar su análisis; los paquetes de una captura van siendo procesados uno a uno para obtener su protocolo y, según sea éste, realizar un ensamblado acorde para obtener los datos y analizarlos a su vez, de forma que se consigue ir subiendo por la torre de protocolos hasta llegar a los datos del nivel de aplicación. Podemos encontrarnos con que el paquete utiliza IPv4 o IPv6, además podría tratarse de un paquete UDP o TCP, HTTP o SMTP, si es HTTP podría ser una petición o una respuesta, etc. Del procesado por protocolos se encargan los llamados manipuladores de datos, cada uno de ellos específicamente diseñado para analizar un protocolo. Esta forma de manejar los datos permite además relacionar los paquetes que pertenecen al mismo flujo y unificar los datos que inicialmente estaban repartidos, obteniendo la información que el usuario ha recibido, visualizado y enviado, como por ejemplo, una página web completa o un correo electrónico. Es precisamente el proceso inverso al de creación del paquete. Los contenidos decodificados son entonces almacenados bajo su sesión correspondiente, según su tipo: página web, *email*, *webmail* (correo online), documentos pdf, documentos impresos, conexiones telnet o ftp, etc. Toda la información de las sesiones se almacena bajo su caso y los casos se almacenan a su vez bajo el árbol de directorios de la herramienta. Además *Xplico* utiliza una base de datos que almacena las referencias a los contenidos decodificados junto con información obtenida durante el análisis.

Es necesario indicar que la herramienta presenta limitaciones de cara a utilizarla para la interceptación legal de comunicaciones. Probablemente la más clara es que, ya que cada protocolo requiere un procesamiento diferente, es necesario tener tantos manipuladores como protocolos. Esto se va solventando poco a poco según se van añadiendo nuevos manipuladores y mejorando los existentes; la Figura 11 muestra el estado actual de mismos. En el caso de la navegación por Internet, *Xplico* se ve afectado por la caché de los navegadores ya que decodifica la información presente en el flujo de

tráfico que recibe, por lo que si por ejemplo el navegador ya tenía almacenadas las imágenes de una página web, en las peticiones que realice al servidor no las solicitará por lo que la página web decodificada sólo mostrará que debería haber imágenes pero no cuáles. Sin embargo, la mayor limitación en la decodificación de datos es que estos estén cifrados; en ese caso, *Xplico* simplemente no es capaz de obtener la información.

Dissector	Status	Note
ARP	90%	—
Radiotap	90%	—
Ethernet	100%	—
PPP	90%	—
VLAN	95%	—
L2TP	70%	—
IPv4	98%	—
IPv6	98%	—
TCP	95%	—
UDP	100%	—
DNS	80%	—
HTTP	100%	—
SMTP	95%	—
POP	95%	—
IMAP	95%	—
SIP	80%	—
MGCP	85%	—
H323	5%	—
RTP	80%	—
RTCP	75%	—
SDP	70%	—
FB chat	90%	—
FTP	90%	—
IPP	90%	—
CHDLC	80%	—

Dissector	Status	Note
PJL	90%	—
NNTP	95%	—
MSN	60%	v1 beta
IRC	85%	—
YAHOO	0%	—
GTALK	0%	—
EMULE	0%	—
SSL/TLS	0%	with keys
IPsec	0%	with keys
802.11	60%	no encryp.
LLC	60%	—
MMSE	95%	over HTTP
Linux cooked	95%	SLL
TFTP	90%	—
SNOOP	100%	Format
PPPoE	90%	—
Telnet	90%	—
WebMail	90%	—
Paltalk Exp.	60%	—
Paltalk	90%	—
NetBIOS	5%	Ses. Mes.
SMB	0%	—
PPI	90%	—
syslog	100%	—
–		—

Figura 11. Estado de los manipuladores de datos de Xplico [4].

Una vez explicado el funcionamiento general de *Xplico*, veamos su utilización. La interfaz gráfica es una interfaz web que interactúa con la base de datos y permite tanto crear nuevos casos y sesiones como visualizar los ya existentes. En la Figura 12 se observa la pantalla inicial que se muestra al seleccionar visualizar una sesión concreta. En la parte de arriba hay un menú que permite acceder a sitios web relacionados con *Xplico* (Ayuda, Foro, Wiki, Licencias) y cambiar la contraseña del usuario. En la parte izquierda se encuentra el menú de navegación. La primera opción *Cases* es común a todas las pantallas y permite crear nuevos casos y/o sesiones y moverse entre los casos y sesiones existentes. El resto de opciones sólo están presentes durante la visualización de sesiones y permiten acceder a los contenidos y su información. La parte central muestra los detalles de la sesión. Arriba a la izquierda aparecen el nombre del caso y de la sesión, la hora en que empezó y terminó la decodificación y su estado. Esta pantalla se corresponde a una sesión de análisis de ficheros por lo que arriba a la derecha se permite cargar un fichero *pcap*; si se tratara de una sesión de captura en tiempo real, las opciones permitirían iniciar, parar y detener la captura. El resto de la pantalla muestra información sobre los contenidos decodificados bajo la sesión.

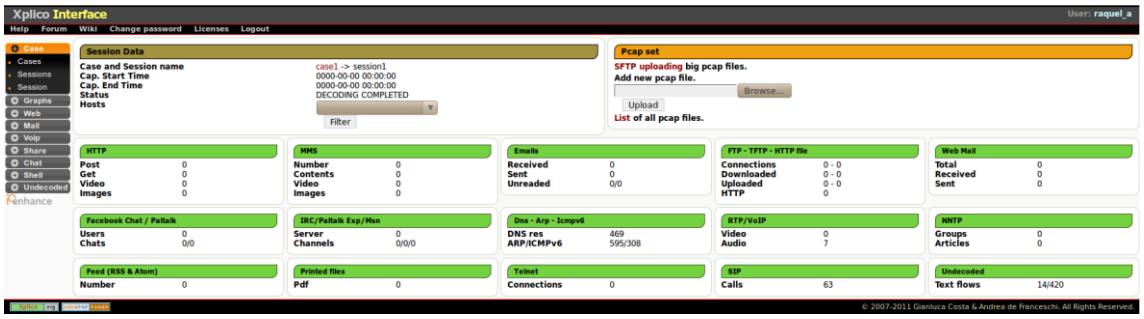


Figura 12. Vista de la interfaz web de Xplico: organización de los contenidos por tipo.

Al tratarse de una interfaz web, para acceder tan sólo es necesario abrir un navegador y teclear *http://nombre_máquina:9876*, donde en nuestro caso *nombre_máquina* es el nombre o dirección IP de la máquina en que se realizó la instalación de *Xplico* ya que no vamos a utilizar el servicio DNS. La pantalla inicial permite escoger el idioma y requiere al usuario que se identifique para acceder al sistema mediante un nombre de usuario y contraseña. Estos datos entonces se contrastan con los almacenados en la base de datos de la aplicación para comprobar si son correctos y, en caso afirmativo, si se trata del administrador del sistema o un usuario. Ésta forma de acceder a la plataforma será una de las primeras modificaciones a realizar puesto que no es lo suficientemente segura para una plataforma de LI. En su lugar se va a permitir el uso de certificados digitales que autentiquen a los usuarios, tal como se explica en la sección 3.2.

Capitulo 3: **Diseño e implementación**

En los siguientes apartados se explicarán las decisiones que se han tomado a la hora de crear el sistema decodificador a partir de *Xplico* mediante la integración de cada uno de los módulos desarrollados. La Figura 13 muestra el esquema de la estación decodificadora de ILIP integrado en el sistema de INDECT. Un analista se conecta bien al servidor de servicios INDECT o bien directamente al servidor del servicio. En ambos casos se le autentica mediante certificados digitales contra un servidor LDAP (*Lightweight Directory Access Protocol*). El servidor del servicio ILIP contiene tres tipos de elementos: la aplicación web del servicio, la estación decodificadora de ILIP propiamente dicha y los *plugins* locales de la misma. Además existe un cuarto tipo de servidor que almacena *plugins* remotos. En la Figura 13 sólo se incluye un servidor LDAP y uno de *plugins* remotos, pero pueden aparecer tantos como sea necesario.

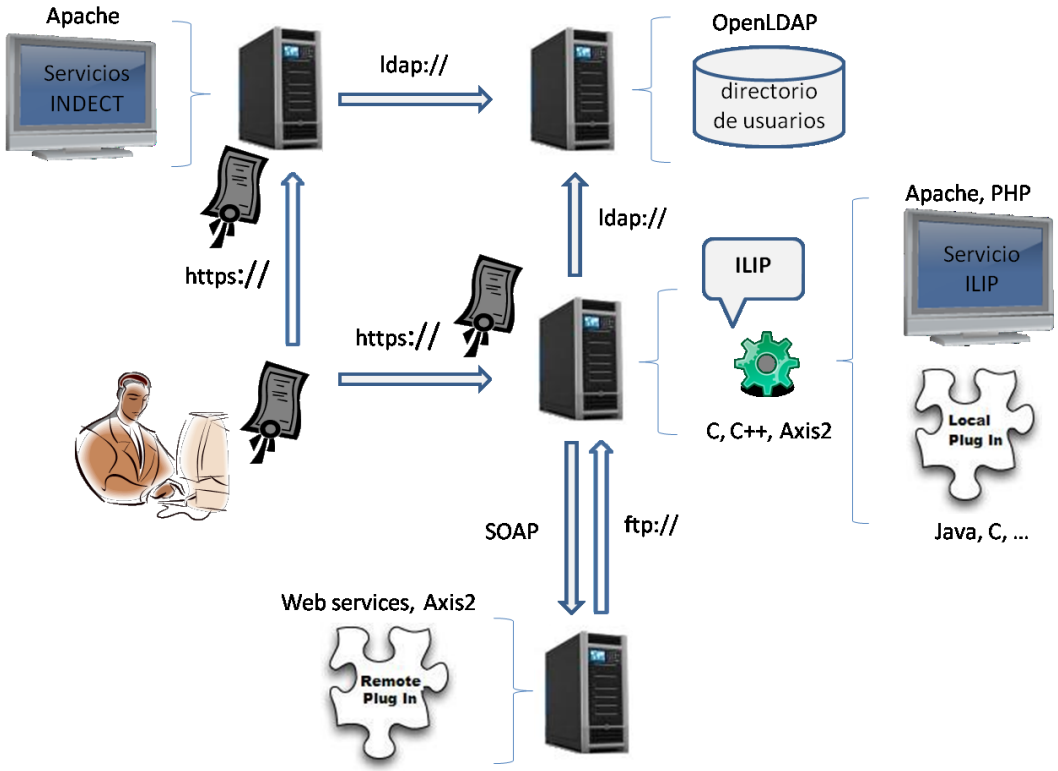


Figura 13. Sistema INDECT completo con la estación decodificadora integrada.

En el apartado **3.1** se entrará más en detalle en el funcionamiento de *Xplico*, para así poder entender las decisiones tomadas en los apartados siguientes.

La seguridad de la interfaz web se incrementó mediante el uso del protocolo HTTPS (*Hyper Text Transfer Protocol Secure*), una PKI (*Public Key Infrastructure*), y un servidor LDAP. Esto se explicará por separado en dos apartados. En el **3.2** se explicará la PKI desplegada, las autoridades de certificación, los certificados digitales y la configuración del sistema. En el **3.2** se definirá el árbol de directorio LDAP que va a almacenar la información de los usuarios del sistema y cómo hacer que la plataforma lo utilice.

A continuación se presenta un ejemplo de integración con una herramienta ya existente, LINK, en el apartado **3.4**.

Hasta ahora, todos estos pasos solamente requieren la modificación de la interfaz web. En los que vienen a continuación es necesario modificar el propio código de la herramienta. La integración de los módulos se basa en los siguientes criterios:

- El módulo distribuidor de contenidos se invoca cuando se tienen todos los datos necesarios para procesar un contenido. Éste módulo se encarga de enviar dichos datos y el contenido a unas aplicaciones denominadas *plugins* de preclasificación.
- El módulo de transcripción VoIP-texto se invoca cuando un contenido de audio ha sido obtenido del protocolo SIP o RTP, para generar un contenido de tipo texto que a su vez sea enviado al distribuidor.
- El módulo de identificación de contenidos mediante hashes difusos se integrará como un *plugin* remoto puesto que mucho más cómodo y eficiente para mantener una lista centralizada de contenidos. Este paso no se ha realizado aún.

Primero se hablará de los *plugins* de preclasificación en el apartado **3.5** y a continuación se procederá a explicar el módulo distribuidor de contenidos en el apartado **3.6**. Por último, la integración del módulo de transcripción VOIP-texto se explica en el apartado **3.7**.

3.1 *Xplico* en detalle

La Figura 14 muestra la relación entre los cuatro macrocomponentes que componen *Xplico*:

- Un gestor global llamado *DEMA* (*decoder manager*).
- Un decodificador de tráfico de red (generalmente también denominado *Xplico*, ya que es la base de la herramienta).
- Un conjunto de manipuladores de datos.
- Un interfaz gráfico para la visualización de los datos.

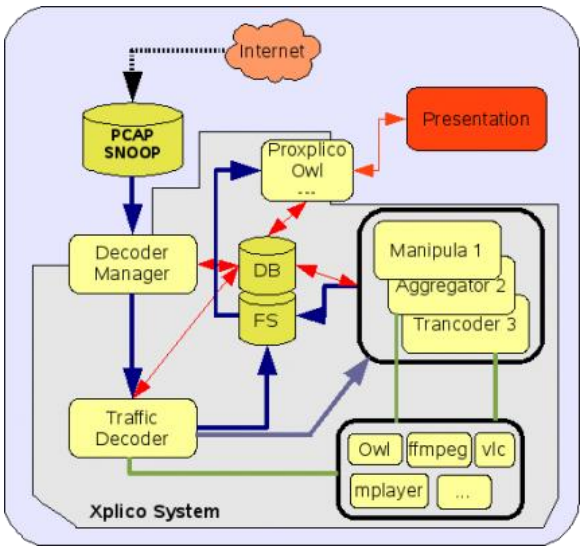


Figura 14. Componentes de la herramienta Xplico [4].

El gestor DEMA se encarga de organizar los datos de entrada, realizar la configuración del decodificador y los manipuladores según unos ficheros de configuración, así como lanzarles y controlar su ejecución. Son estos dos últimos componentes los que realizan la decodificación propiamente dicha. El decodificador de tráfico de red recibe el tráfico de entrada obtenido por los módulos de captura (*capture modules*) que son interfaces a cualquier tipo de sistema de adquisición. Existen dos módulos de captura, uno que permite manejar ficheros con formato *pcap* (utilizado por tcpdump y wireshark) y otro, denominado *snoop*, que permite capturar tráfico en tiempo real. La estación decodificadora no necesita hacer uso del segundo ya que será la estación de captura la que obtenga el tráfico a analizar y lo almacene en ficheros *pcap*. La información recibida pasa a los llamados módulos de procesamiento (*dissector modules*), encargados de procesarla según su protocolo. La Figura 15 muestra algunos ejemplos. Por último, los contenidos decodificados son enviados a la salida seleccionada mediante los módulos de distribución (*dispatcher modules*) que son interfaces a cualquier tipo de sistema de almacenamiento de datos. *Xplico* proporciona los siguientes módulos de distribución:

- *cli* almacena los datos en el directorio *xdecode/* con un directorio separado para cada IP origen y a su vez organizado como un directorio por protocolo.
- *lite* almacena los datos en una base de datos de tipo SQLite.
- *none* imprime por pantalla parte de la información decodificada sin almacenarla.

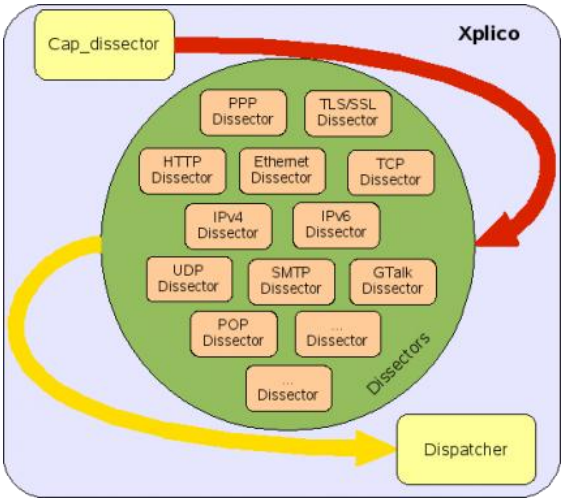


Figura 15. Módulos de procesamiento de Xplico [4].

En la estación decodificadora se utilizará el módulo de distribución *lite*. Este módulo proporciona el punto adecuado para realizar las llamadas a los distintos módulos, ya que se tiene acceso a toda la información necesaria acerca de cada contenido, por lo que es a través de él donde se realiza la integración software. Veamos cómo está organizada la base de datos. Ésta contiene una o incluso dos tablas por tipo de contenido, como es el caso de los *chats* de *facebook* donde una tabla almacena los usuarios y otra tabla almacena los *chats* en sí. Las tablas se nombran con el nombre del tipo de contenido en plural, por ejemplo los mensajes ARP se almacenan en la tabla *arps* y los MMS en la tabla *mmss*. En el caso de múltiples tablas para un mismo tipo de contenido, el nombre de la tabla hija suele ser más específico como es el caso de la tabla padre *mms* y la tabla hija *mms_contents* aunque también hay casos en que ambas tienen nombres más descriptivos tal y como vemos en las tablas *nntp_articles* y *nntp_groups*. Algunos

ejemplos de tipos de contenido son: *arp*, *dns*, *fbwchat* (*chat* de *facebook*), *ftp* y *tftp* (descarga de ficheros), *http*, *icmpv6*, *mms*, *msn* (*messenger*) y *webmsn* (versión online), *rtp y sip* (ambos para VoIP), *telnet*, *mail* (correo tipo SMTP o POP) *y webmail* (correo *online* tipo Hotmail, Yahoo, Gmail),…

Volviendo a la integración, es necesario modificar la base de datos para que almacene la información que proporcionan los nuevos módulos. Por ejemplo:

- Los *plugins* proporcionan la preclasificación en forma de un resultado numérico y una descripción textual, por lo que es necesario añadir estos dos campos a todas las tablas de la base de datos que almacenan contenidos decodificados.
- El módulo distribuidor utiliza una base de datos propia de *plugins* y reglas que hay que integrar en la de *Xplico* para crear una base de datos única para la plataforma decodificadora.
- El módulo de conversión VoIP-texto genera documentos de texto a partir de los ficheros de audio obtenidos de la decodificación de conversaciones. Por tanto es necesario añadir algunas tablas nuevas para indicar dónde están almacenados dichos documentos y cuándo se realizó la conversión. Además es útil añadir en las tablas de *Xplico* que almacenan la información sobre las conversaciones VoIP, campos que almacenen referencias a la nueva tabla.

Algunos de estos cambios en la base de datos han de verse reflejados también en la interfaz gráfica, en la que además es necesario realizar modificaciones que permitan utilizar los certificados digitales de usuario y comprobar las autorizaciones contra un servidor LDAP en lugar de simplemente en la base de datos de la aplicación. Ésta interfaz gráfica es una interfaz web diseñada bajo CakePHP [28] que utiliza el lenguaje PHP y el patrón Modelo-Vista-Controlador, como se observa en la Figura 16:

- El modelo es la representación específica de la información que el sistema maneja: los datos de la aplicación. Como ya se ha mencionado, *Xplico* ha sido configurado para hacer uso de una base de datos de tipo SQLite.
- La vista es la presentación de los datos: la interfaz de usuario.
- El controlador es la lógica de la aplicación y responde a eventos (acciones del usuario) e invoca peticiones al modelo y a la vista.

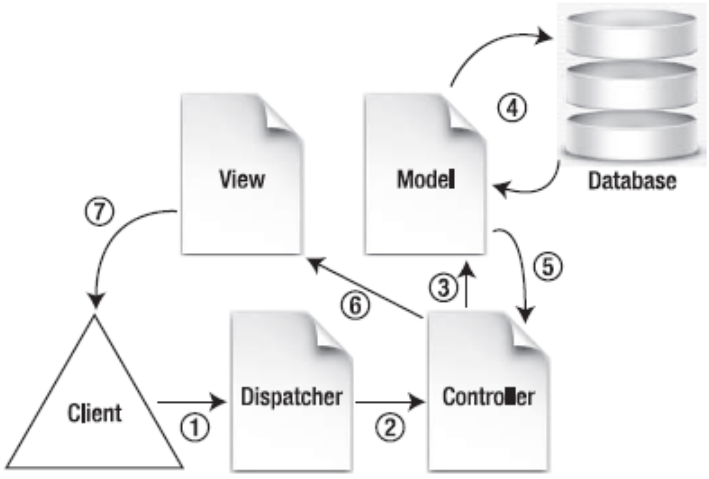


Figura 16. Patrón Modelo-Vista-Controlador en CakePHP [29].

El modelo está compuesto por ficheros con extensión *.php* que indican información sobre las tablas de la base de datos, incluídas las relaciones con otras tablas. La vista está compuesta por ficheros *.ctp* que pueden combinar lenguajes como HTML, PHP, Javascript para generar el contenido a enviar al usuario, ya sea éste una página web o un fichero para descargar. El controlador está compuesto por ficheros *.php* que contienen funciones que tramitan las solicitudes y ayudan a coordinar y preparar la respuesta para el cliente. Esta información será suficiente para poder explicar los cambios realizados. En cualquier caso, el **Anexo I** contiene más información sobre las bases de este entorno de trabajo.

Ahora se procederá a comentar cómo está organizado el sistema de ficheros de *Xplico* en general y el subsistema correspondiente a la aplicación web en particular. Las modificaciones que sea necesario realizar se irán viendo posteriormente en sus correspondientes apartados. *Xplico* queda instalado bajo el directorio */opt/xplico*, que contiene principalmente la base de datos denominada *xplico.db* y los siguientes subdirectorios:

- *bin/*: contiene librerías y ejecutables.
- *cfg/*: contiene ficheros de configuración de la aplicación.
- *log/*: directorio para almacenar los registros de la aplicación.
- *script/*: contiene scripts.
- *xi/*: subdirectorio de la interfaz web.

Además de los anteriores, se generará un directorio *pol_X/* por cada caso almacenado, donde *X* es el identificador numérico del caso. Dentro de él encontramos varios directorios, de los que nos interesan los del tipo *sol_Y*, que almacena la información correspondiente a la sesión *Y*, donde *Y* es el identificador numérico de la sesión. Bajo los directorios de este tipo aparecen una serie de carpetas que almacenan los contenidos decodificados y que se corresponden con las tablas almacenadas en la base de datos: *arp, dns, fbwchat, ftp, http, icmpv6, mail, mms, msn, rtp, sip, telnet, tftp, webmail, webmsn* y otros.

El directorio *xi/* se corresponde con el directorio estándar de CakePHP. De entre todos los directorios que contiene nos interesa *app/*, bajo el que se encuentran entre otras:

- Una carpeta *controllers/* que almacena los ficheros *.php* que contienen los controladores.
- Una carpeta *models/* que almacena los ficheros *.php* que contienen los modelos.
- Una carpeta *views/* que almacena los ficheros *.ctp* que contienen las vistas.
- Una carpeta *webroot/* que almacena imágenes u otros ficheros para incluir en las vistas, scripts en código javascript, ficheros *.css* que definen estilos para las vistas, etc.
- Una carpeta *config/* que almacena la configuración de la aplicación web, como por ejemplo el fichero *database.php*, que contiene la información de la base de datos a utilizar: tipo, usuario y contraseña, localización, etc.

Son estos directorios los que se modificarán para realizar las distintas integraciones en la interfaz web, en algunos casos modificando ficheros ya existentes y en otros

añadiendo nuevos ficheros. Este último es el caso del módulo distribuidor de contenidos o del módulo de transcripción VoIP-texto, puesto que ambos incluyen una interfaz web que es necesario unificar con la de *Xplico*.

3.2 Incrementando la seguridad del sistema: HTTPS y certificados digitales

Dado que la estación decodificadora de la plataforma ILIP está basada sobre una aplicación web ya existente, uno de los principales puntos de partida ha sido comprobar y aumentar su seguridad. El protocolo HTTP es inseguro ya que envía la información entre servidor y cliente sin cifrar. Por el contrario, el protocolo HTTPS crea un canal cifrado basado en SSL/TLS en el puerto 443 para evitar que un atacante pueda interceptar la transferencia de datos de la conexión entre el cliente y el servidor. Esto es lo que se denomina confidencialidad y es uno de los cuatro objetivos de los algoritmos criptográficos:

- Confidencialidad: garantiza el acceso únicamente a usuarios autorizados. Se usan códigos y técnicas de cifrado.
- Integridad: garantiza la corrección y completitud de la información. Por ejemplo, se usan funciones hash criptográficas MDC.
- No repudio: garantiza que no se pueda negar la participación. Por ejemplo, se usa la firma digital.
- Autenticación: garantiza la identidad del comunicante. Por ejemplo, se usan funciones hash criptográficas MAC.

Los dos tipos básicos de cifrado que existen son el simétrico y el asimétrico, según si la clave usada para cifrar es la misma que la usada para descifrar o no. Algunos algoritmos simétricos de bloque son DES, 3-DES, RC2, RC5, RC6 y Rijndael o AES y entre los asimétricos se encuentran Diffie-Hellman, RSA, DSA o ECC. Los algoritmos criptográficos simétricos cifran y descifran con la misma clave. Pueden ser de bloque si manejan pequeños bloques de datos o de flujo si manejan un bit cada vez. El algoritmo simétrico de flujo más usado es RC4. Por otro lado, los algoritmos asimétricos utilizan dos claves matemáticamente relacionadas entre sí, una pública y una privada, que tienen la propiedad de descifrar lo que la otra ha cifrado pero no lo que ha sido cifrado por ellas mismas. La clave pública se da a conocer a todo el mundo mientras que la privada se mantiene lo más segura posible de manera que:

- si se usa la clave pública de otra entidad para cifrar un mensaje dirigido a ella, sólo ésta será capaz de descifrarlo usando su clave privada: se asegura la confidencialidad e integridad de los datos. Es el denominado cifrado de clave pública,
- si se usa la clave privada de una entidad para cifrar un mensaje, cualquiera podrá descifrarlo mediante la clave pública de dicha entidad, por lo que, basándose en que la clave privada está guardada de forma segura, se sabe que sólo pudo haber sido esa entidad la que cifró el mensaje: esto permite la identificación y autentificación de la entidad. Es la denominada firma digital.

En cuanto a prestaciones, el cifrado simétrico es rápido y compacto pero no permite

firmar ni garantiza el no repudio, mientras que el cifrado asimétrico sí que los permite aunque es lento y la información cifrada puede ocupar más que la información en sí. Además, el cifrado simétrico requiere tanto una clave por comunicación, por lo que no escala bien, como el envío de la clave antes de la propia comunicación, por lo que requiere el establecimiento de un contacto inicial y es susceptible de ser interceptada. Esto no ocurre con el cifrado asimétrico, puesto que se utilizan las propiedades de las claves pública y privada. La combinación de ambos tipos de algoritmos proporciona una solución mucho mejor que el uso individual de cualquiera de ellos: para cifrar la información se utiliza una clave simétrica, por lo que el proceso es rápido y compacto. Para proteger dicha clave se utiliza el sistema asimétrico. Se busca la clave pública del destino y se usa para cifrar la clave simétrica. Ya que la clave es de tamaño reducido, el proceso es rápido. Por último, se envían juntas la clave simétrica cifrada con la clave pública y la información cifrada con la clave simétrica. De esta manera, sólo el destino es capaz de descifrar el mensaje, ya que sólo él conoce la clave privada necesaria para descifrar la clave que a su vez descifra la información. Sin embargo, ésto implica confiar en que la clave pública de un usuario corresponde realmente a ese usuario. Para solucionar esto existen varios esquemas y HTTPS utiliza las infraestructuras de clave pública o PKI basadas en certificados X.509 y entidades emisoras de certificados llamadas autoridades de certificación o CAs (*Certification Authorities*) que aseguran la autenticidad de la clave pública y ciertos atributos del usuario, sea este una persona, una compañía, un sitio web,... Una CA es una organización fiable que acepta solicitudes de certificados de entidades, las valida, genera certificados y mantiene la información de su estado. Debe proporcionar una *Declaración de Prácticas de Certificación* (*Certification Practice Statement* o *CPS*) que indique claramente sus políticas y prácticas en cuanto a seguridad y mantenimiento de certificados, sus responsabilidades y las obligaciones de los suscriptores respecto a los sistemas que emplean sus certificados. Las labores básicas de una CA son:

- Admisión de solicitudes: un usuario rellena un formulario de solicitud de certificado y lo envía a la CA. La generación de las dos claves pública y privada puede ser realizada por el usuario o por un sistema asociado a la CA.
- Autentificación del sujeto: antes de firmar la información proporcionada por el sujeto la CA debe verificar su identidad.
- Generación de certificados: tras recibir una solicitud y validar sus datos la CA genera el certificado, lo firma con su clave privada, lo manda al subscriptor y, opcionalmente, lo envía a un almacén de certificados para su distribución.
- Distribución de certificados: la CA puede proporcionar un servicio de distribución de certificados mediante correo electrónico o servicios de directorio como el X.500 o el LDAP, a la que las aplicaciones tengan acceso y de la que puedan obtener los certificados de sus subscriptores.
- Anulación o revocación de certificados: al igual que con las solicitudes de certificados, la CA debe validar una solicitud de revocación. Además debe mantener información sobre una revocación durante todo el tiempo de validez del certificado original.
- Almacenes de datos: la CA almacena toda la información de las solicitudes, claves públicas y certificados en bases de datos.

Las CAs pueden haber sido a su vez validadas por otras CAs, y así sucesivamente

hasta llegar a una CA raíz cuyo certificado es autofirmado, es decir, ella misma firma su propio certificado. Esto introduce los conceptos de cadena de confianza y de jerarquía de CAs. Un certificado se considera válido si la entidad que lo firmó se considera válida, por lo que para validar un certificado hay que usar toda la cadena de CAs desde la que lo validó directamente hasta la raíz; ésta es la cadena de confianza. Dicha cadena puede corresponderse con CAs de distintos niveles en una determinada estructura ordenada denominada jerarquía de CAs. Empezando en una CA raíz autofirmada que se valida a sí misma, la función de cada *CA* de un nivel es validar los certificados de las CAs de nivel inferior hasta llegar a CAs que ya no pueden firmar otras CAs si no sólo certificados finales. Esto permite crear ramificaciones especializadas en algún tipo de certificados: la cadena de validación es diferente según el tipo de certificado. Una jerarquía de CAs puede crearse de forma privada, con la estructura de niveles y los datos en los certificados que cada sistema requiera, de forma que los elementos de dicho sistema confían sólo en los certificados generados por su jerarquía y no por otra. Una jerarquía privada proporciona independencia de las CAs y jerarquías conocidas, no sólo en términos económicos y de administración sino también de seguridad, puesto que la seguridad del sistema se vería comprometida en cuanto una de las CAs fuera vulnerada.

Como conclusión, para poder hacer uso del protocolo HTTPS, el mínimo requisito es que el servidor tenga un certificado de clave pública que presentar a los clientes, que decidirán si confían o no en dicho certificado según la autoridad de certificación que lo haya firmado. Los navegadores web se distribuyen con los certificados raíz de las CAs más típicas para que éstos puedan verificar certificados firmados por ellas. En el caso de que no se conozca a la CA emisora del certificado, como es el caso de las jerarquías privadas, se le indica al usuario para que tome una decisión. De esta manera el usuario se asegura de estar conectándose dónde realmente quiere. Una mejora es proporcionar certificados a los clientes para poder autenticarlos, es decir, el servidor pueda asegurarse de que son quienes dicen ser. En el sistema, en vez de obligar a que los clientes proporcionen un certificado, se ha decidido hacerlo opcional, de manera que si lo tienen, la autenticación es inmediata, y sólo hay que comprobar en el sistema si dicho usuario tiene autorización para lo que ha pedido y, si no tiene un certificado, debe proporcionar unos datos de acceso: un identificador interno (alfanumérico preferentemente) para identificarse y una contraseña para autenticarse, que son empleados para comprobar si está o no autorizado.

Además, los certificados de cliente serán utilizados tanto mediante la importación directa a un navegador como mediante el uso de dispositivos de seguridad. Un dispositivo de seguridad es un dispositivo electrónico de pequeño tamaño utilizado para almacenar claves criptográficas que prueben la identidad de una persona electrónicamente, tales como datos biométricos (por ejemplo una huella digital) o firmas digitales o certificados, como en nuestro caso. Algunos pueden tener un pequeño teclado que permite introducir un PIN (*Personal Identification Number*), un conector USB, funciones RFID o una interfaz Bluetooth para poder grabar, acceder y borrar los datos.

Para generar los certificados de la plataforma ILIP se ha optado por crear la jerarquía de CAs privada mostrada en la Figura 17, generalizable a la mostrada en la Figura 18. Está organizada en tres tipos de CAs: raíz, de usuario y de servidor. El primer tipo es la CA raíz que valida los certificados de los otros dos tipos. Esta CA se asocia con una organización y sólo firma certificados para entidades pertenecientes a dicha organización. En nuestro proyecto la organización es INDECT y la CA se denominará

RootCA. Los otros dos tipos de CA validan certificados de usuario y de servidor, respectivamente, y se denominarán simplemente *UserCA* y *ServerCA*. Por seguridad todas las CAs funcionarán de modo *off-line*, es decir, cuando es necesario validar y firmar un certificado, será una persona autorizada la que acceda a la máquina localmente y realice el proceso.

Según esta estructura, los certificados para clientes y servidores se validan con diferentes cadenas de CAs lo que permite configurar cada elemento del sistema de una forma diferente y asegurarse que un cliente acepte conectarse a un servidor pero no a otro cliente y que un servidor web acepte una conexión procedente de un cliente pero no de otro servidor. Se ha adoptado esta estructura por su sencillez y validez para la plataforma. Se podrían utilizar otras opciones más complejas como por ejemplo que no existiera una *RootCA* sino directamente una *UserCA* y una *ServerCA* que se validaran la una a la otra mediante firmas cruzadas o también se podría hacer distinción entre los distintos tipos de servidores mediante el uso de CAs específicas..

Encontramos dos herramientas principales para la creación de claves, certificados y CAs: OpenSSL y GnuTLS. Se han realizado pruebas con ambas herramientas y la elección ha sido OpenSSL, básicamente porque ha resultado más sencilla de configurar y utilizar y es la que se utiliza en otras partes del proyecto INDECT. El comando openssl es una utilidad criptográfica que implementa los protocolos de red SSL y TLS y los estándares relacionados con éstos.

Para no saturar este apartado con información técnica se ha creado el **Anexo II**, que contiene información sobre los certificados y el proceso a llevar a cabo para cada paso:

- *Certificados X.509.*
- *Formato del fichero de configuración de la herramienta OpenSSL.*
- *Creación de una jerarquía de autoridades de certificación mediante OpensSSL.*
- *Uso de los certificados digitales de cliente. Dispositivos de seguridad.*
- *Uso de los certificados digitales de servidor. Instalación de un servidor web seguros.*

Aquí simplemente se esbozará el proceso y se comentará la configuración necesaria.

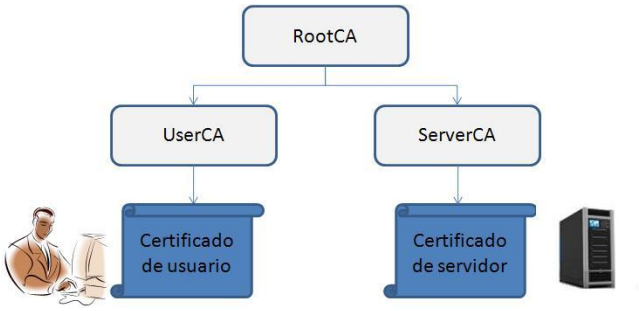


Figura 17. Jerarquía de autoridades de certificación para la plataforma ILIP.

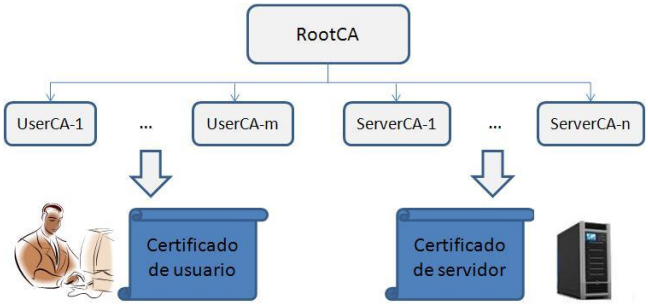


Figura 18. Jerarquía genérica de autoridades de certificación.

Un certificado tiene una serie de campos obligatorios, entre ellos el campo *Subject*. Este campo contiene la identidad cuya clave pública está certificada en forma de una cadena compuesta jerárquicamente por un conjunto de atributos y sus valores. En nuestra PKI los atributos a incluir dependerán de a quién se va a certificar: *RootCA*, *ServerCA*, *UserCA*, usuario o servidor. Los siguientes campos son comunes a los cinco tipos de certificados:

- *countryName*: país.
- *stateOrProvinceName*: provincia.
- *organizationName*: organización.
- *organizationalUnitName*: departamento dentro de la organización.
- *commonName*: nombre común.
- *emailAddress:* dirección de correo electrónico

Los certificados correspondientes a entidades finales (servidor o usuario) añaden los siguientes campos:

- *localityName*: localidad.

Por último, los certificados de usuario incluyen un campo que identifica unívocamente al usuario dentro del sistema:

- *userId*: identificador del usuario.

Una vez definidos los campos de los certificados ya es posible definir la configuración de las CAs:

- *RootCA*:
  - Firma certificados para CAs con una duración de 5 años.
  - El periodo de tiempo entre de listas de revocaciones es de 30 días.
  - El uso de los certificados firmados se restringe a firma de certificados y de revocaciones.
- *ServerCA*:
  - No firma certificados para CAs sino sólo para entidades servidor, con una duración de 1 año.
  - El periodo de tiempo entre de listas de revocaciones es de 7 días.
  - El uso de los certificados firmados se restringe a firma digital y cifrado.
- *UserCA*:
  - No firma certificados para CAs sino sólo para entidades de tipo usuario, con

una duración de 3 años.

- El periodo de tiempo entre listas de revocaciones es de 30 días.
- El uso de los certificados firma dos se restringe a firma digital, cifrado y no-repudio.

Por motivos de seguridad, en los tres casos se ha elegido como algoritmo para la generación del *message digest* (resumen de mensaje) el sha256. La longitud de la clave privada es de 4096 bits para los certificados de CAs, que además la protegen mediante cifrado AES de 256 bits, 2048 bits para los de tipo servidor y 1024 bits para los de tipo usuario. La diferencia es debida a los distintos niveles de seguridad en que se encuentra cada uno. Es vital que las CAs no se vean comprometidas puesto que eso podría en peligro todo el sistema. El caso de un usuario final es el caso más sencillo de tratar, sin más necesidad que revocar su certificado. Sin embargo si el certificado de un servidor se viera comprometido los riesgos serían mucho mayores.

En cuanto a la estructura de ficheros que almacenará la clave privada de un elemento cualquiera (CA, servidor o usuario) y su correspondiente certificado se va a utilizar la que propone OpenSSL. En cualquiera de los casos, la clave privada es un dato que se debe almacenar de forma muy segura, tanto con técnicas de cifrado como mediante el uso de permisos muy restrictivos. Si se almacena información de un usuario o servidor, típicamente basta con dos directorios: uno para almacenar los certificados públicos (*certs*) y por lo tanto accesible por otros y otro para la clave privada (*private*). Sin embargo, añadiremos dos directorios más, uno que almacene las peticiones de certificados (*csrs*) y otro que almacene los certificados revocados (*crl*), como se observa en la Figura 19 para el usuario Raquel Aparicio.

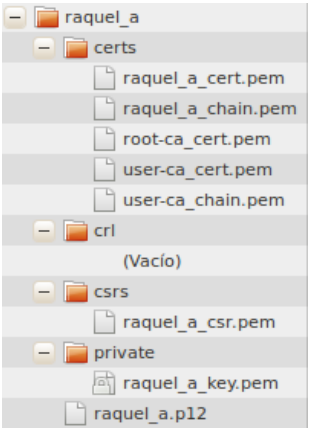


Figura 19. Estructura de ficheros del usuario Raquel Aparicio (raquel_a).

Por el contrario, una CA tiene mucha más información que mantener: certificados pendientes de firma, firmados o revocados, por lo que es interesante mantenerlos en directorios separados. La Figura 20 muestra como ejemplo la estructura de directorios elegida para la CA raíz, tras haber firmado dos certificados, uno para una *ServerCA* y otro para una *UserCA*, donde:

- *certs*: almacena las claves y los certificados públicos propios de la CA, en este caso, el certificado raíz autofirmado *root-ca_cert.pem*. Al tratarse de los datos públicos, este directorio es accesible por otros.

- *crl*: almacena las listas de revocación de certificados (*Certificate Revocation Lists*).
- *csrs*: almacena las peticiones de certificación (*_csr*) y los certificados obtenidos al firmarlas (*_cert*).
- *newcerts*: almacena los nuevos certificados renombrados a nn.pem donde nn representa el índice en la base de datos de la CA.
- *private*: almacena la clave privada *root-ca_key.pem* por lo que este directorio debe estar muy protegido por permisos de acceso.
- *crlnumber*: almacena el siguiente número para la revocación de certificados.
- *index.txt*: fichero ASCII que almacena la lista de certificados generados.
- *root-ca.cnf*: fichero de configuración para la generación de certificados.
- *serial*: almacena el siguiente número para la creación de certificados.

En teoría, cada CA debería funcionar sobre una máquina diferente bajo fuertes requisitos de seguridad que prevengan que la jerarquía pueda verse comprometida y cada servidor y cada usuario almacenaría su propia información. Sin embargo, de nuevo por simplicidad y debido a que de momento sólo se dispone de dos máquinas para desplegar el sistema, se ha elegido mantener la estructura de las tres CAs en la misma máquina.

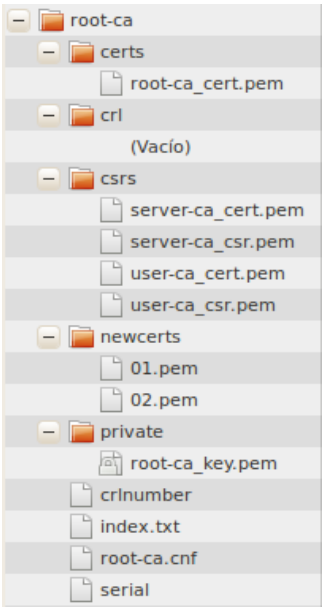


Figura 20. Estructura de ficheros de la autoridad de certificación raíz RootCA.

A continuación se resumen los pasos para crear una CA cualquiera:

1. Crear la estructura de ficheros que almacenará la información de la CA.
2. Crear la base de datos (*index.txt*, *serial* y *crlnumber*).
3. Crear el fichero de configuración para los certificados que genere.
4. Generar un par de claves privada y pública
5. Obtener el certificado propiamente dicho:

5.1. En el caso de una CA raíz, generar un certificado autofirmado.

5.2. Si se trata de una CA subordinada: generar una petición de firma de certificado y enviarla a la CA de nivel superior para que nos devuelva el certificado firmado y la cadena de certificados para validarlo.

Una vez *RootCA*, *UserCA* y ServerCA han sido creadas, vamos a ver los pasos a realizar para obtener un certificado ya sea de usuario o de servidor:

1. Crear la estructura de ficheros que almacenará la información como se muestra en la Figura 21 y Figura 22: directorios *certs. crl, csrs y private*.
2. Generar un par de claves privada y pública.
3. Generar una petición de firma de certificado.
4. Enviarlo a la CA correspondiente para que nos devuelva el certificado firmado y la cadena de certificados para validarlo.
5. En el caso de certificados de usuario, es necesario obtener un fichero en formato PKCS#12 de extensión *.p12* o *.pfx*, empleado para almacenar juntas la clave privada y su certificado X.509 y que permite, por ejemplo, usar los certificados de usuario en los navegadores.

El resultado se muestra en la Figura 21 para un servidor web *ilipserver* y en la Figura 22 para un usuario Raquel Aparicio con identificador raquel_a.

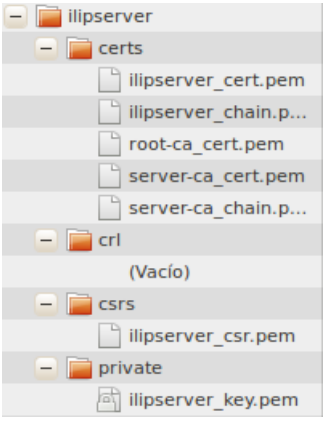


Figura 21. Estructura para almacenar los certificados y claves de un servidor.

Una vez obtenidos los certificados, los pasos para configurar el sistema son muy sencillos. Por un lado, el usuario puede importar su certificado digital en formato PKCS#12 directamente al navegador que utilice o puede almacenarlo en algún dispositivo de seguridad. En nuestro caso emplearemos tanto tarjetas inteligentes como *tokens* USB, concretamente utilizamos los dispositivos de la marca FEITIAN mostrados en la Figura 23: dos tarjetas inteligentes y dos tarjetas tipo SIM, un lector/grabador para las tarjetas de tamaño normal y un *token* USB, lector/grabador para las de tipo *SIM*. Los pasos necesarios para poder emplear este tipo de dispositivos se explican también en el **Anexo II.**

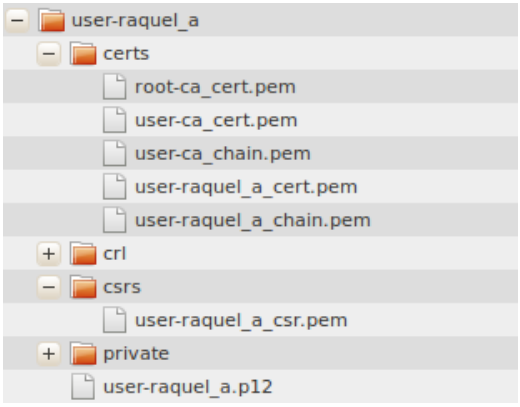


Figura 22. Estructura para almacenar los certificados y claves de un usuario.

En el lado del servidor, hay que tener en cuenta que la aplicación web ésta está implementada sobre el servidor web Apache, cuya configuración es totalmente modular. Consta de una sección *core* sobre la que se van activando otros módulos según se requiera. En el caso de HTTPS hay que activar el módulo *mod_ssl*, que permite mantener comunicaciones seguras usando los protocolos SSL (*Secure Sockets Layer*) y TLS (*Transport Layer Security*). Una vez activado, debemos modificar dos ficheros. El primero es el fichero */etc/apache2/ports.conf* de Apache, donde se indican los puertos sobre los que hay aplicaciones y por lo tanto el servidor debe escuchar. Anteriormente *Xplico* funcionaba sobre el puerto 9876 pero ahora que se utiliza el protocolo *https:* se emplea el puerto seguro 443.

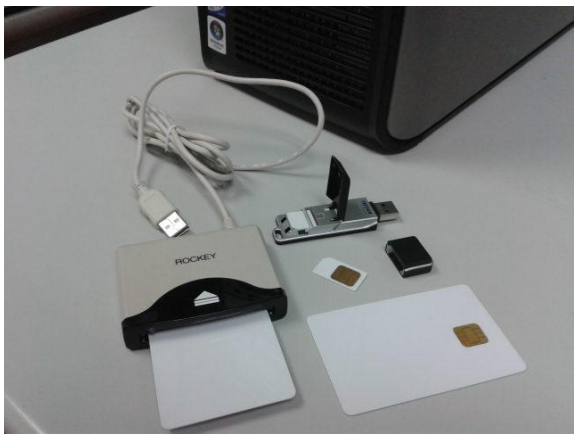


Figura 23. Dispositivos de seguridad empleados en el proyecto.

El segundo fichero es el de configuración del sitio web, */etc/apache2/sites-available/ilip*, al que hay que añadir ciertas directivas que permiten al servidor encontrar la información necesaria sobre los certificados y claves a utilizar. Por motivos de prácticos y de seguridad, se han añadido también otras directivas que realizan una redirección de HTTP a HTTPS, de manera que cualquier intento de conexión *http* en el puerto por defecto pase a estar securizado y controlado automáticamente.

3.3 Incrementando la seguridad del sistema: LDAP

La plataforma ILIP va a formar parte de un sistema mayor. La idea es que el usuario entre primero en dicho sistema a través de su interfaz web y desde ahí ya pueda acceder a la plataforma. El objetivo final es además establecer un procedimiento centralizado de

autenticación y autorización denominado *Single Sign-On*, de manera que los usuarios puedan acceder a las distintas aplicaciones autenticándose una única vez, aunque las aplicaciones mantengan el acceso al sistema de autenticación por si fuera necesario emplearlo. La Figura 13 muestra la relación entre los distintos elementos del sistema. Se observa que las conexiones mediante HTTPS son únicamente las que provienen del usuario ya que el resto del sistema se encuentra protegido bajo grandes requisitos de seguridad, probablemente en un mismo *data center*. El proceso es el siguiente. El usuario se conecta de forma segura a la aplicación web general de los servicios INDECT, que le autentica contra un servidor global de tipo LDAP. Posteriormente el usuario se conecta de nuevo de forma segura a una de las aplicaciones, en este caso a la plataforma ILIP, que comprueba de nuevo al usuario si así se considera necesario. Sólo es posible acceder desde la interfaz web a la estación decodificadora, al menos de momento la estación capturadora no es accesible de esta manera. Además de la aplicación en sí y la interfaz web para acceder a los contenidos, en la misma máquina se encuentran los denominados *plugins* locales, que se verán en su correspondiente apartado. Dentro del sistema también existen los *plugins* remotos, que son los situados en máquinas distintas de que contiene el servidor ILIP y a las que se accede mediante *web services.* Ya que la integración en el sistema de servicios INDECT no es aún posible, de momento simplemente se ha modificado la autenticación en la plataforma para utilizar un servidor LDAP propio.

LDAP (*Lightweight Directory Access Protocol*) es un protocolo de nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuído para buscar información en un entorno de red. Las principales diferencias de un servicio de directorio respecto a una base de datos relacional es la organización jerárquica, con información más descriptiva y basada en atributos, y que su principal objetivo son las búsquedas eficientes ya que la lectura de información se produce con mayor frecuencia que la escritura. Es por ello que se ha elegido esta tecnología para realizar la autenticación del sistema. Para importar y exportar información entre servidores o para describir los cambios a aplicar sobre un directorio, se usan ficheros en formato LDIF (*LDAP Interchange Format*) que contiene directivas que indican la operación a realizar y los datos necesarios. Se puede consultar el **Anexo III** para encontrar información más detallada sobre ambos.

Hay muchas herramientas para crear servicios de directorio propietarias pero en la línea de software libre de código abierto, se encuentran OpenLDAP, Kerberos o Samba. Se ha decidido hacer uso de la herramienta OpenLDAP, que se incluye en muchas distribuciones GNU/Linux pero también en plataformas BSD, Mac OS X, Solaris o algunas Microsoft Windows. OpenLDAP posee tres componentes principales: *slapd* (*Standalone LDAP Daemon*), las bibliotecas que implementan el protocolo y una serie de programas cliente como *ldapsearch, ldapadd, ldapdelete,...* La arquitectura del servidor OpenLDAP se divide en una sección frontal denominada *frontend* que maneja las conexiones de redes y el procesamiento del protocolo y una base de datos de segundo plano llamada *backend* que únicamente maneja el almacenamiento de datos. De esta manera se obtiene una arquitectura modular con una variedad de *backends* para interactuar con diferentes tecnologías. Generalmente una petición LDAP se recibe en el *frontend*, se decodifica y transfiere a uno de los *backends*, que completa la petición y devuelve el resultado al *frontend*, que lo envía al cliente LDAP que realizó la petición. Es posible añadir funcionalidades mediante *overlays*, que consisten en piezas de código insertadas entre estas dos secciones y que se ejecutan bien sobre las peticiones o bien

sobre los resultados.

Dado que aún no se tiene suficiente información sobre la estructura del directorio a utilizar, de momento se ha desarrollado uno muy sencillo. De cada usuario se debe almacenar básicamente información de autenticación e información acerca de las autorizaciones que el usuario posee, es decir, sus permisos en las aplicaciones del sistema. La información de autenticación típicamente consta de un identificador de usuario y una prueba de autenticación que sólo el usuario debería conocer y que, por lo tanto, prueba su identidad. Esta prueba de autenticación no es necesaria en el caso de utilizar un certificado digital de cliente, puesto que el propio certificado es prueba suficiente. Para implementar el sistema no es necesario almacenar información de los certificados en el directorio LDAP pero sí lo es incluir en los certificados el identificador de usuario. Los certificados X.509 permiten incluir en el campo *Subject* un elemento denominado *userId*, cuya función es precisamente la requerida. En términos de seguridad, es preferible utilizar un identificador numérico que haya sido generado dentro del sistema como por ejemplo es el caso del NIA dentro de la universidad. En resumen, el sistema almacena para cada usuario un identificador *userId* y una contraseña. Si el usuario no presenta un certificado digital, debe introducir su contraseña para autenticarse en el sistema. Por el contrario, si el usuario presenta un certificado, éste por sí mismo proporciona la autenticación por lo que sólo es necesario comprobar si dicho usuario está en el sistema mediante la búsqueda de su identificador en el directorio, sin necesidad de contraseña.

En cuanto a la autorización, el directorio debe almacenar a qué aplicaciones tiene acceso cada usuario e indicar sus permisos en la misma. Dos soluciones totalmente opuestas serían o mantener dicha información en la base de datos de la propia aplicación, por lo que el servidor LDAP funcionaría únicamente como un servidor de autenticación, o realizar ambas directamente en el servidor LDAP, por lo que la aplicación queda liberada de dichas comprobaciones. Para mantener la centralización que proporciona el directorio LDAP, se ha optado por la última. El directorio almacena información de las aplicaciones bajo la entrada *applications*, de los usuarios bajo la entrada *users* y los permisos de cada usuario en cada aplicación bajo la entrada del propio usuario. Esto se observa en la Figura 24, cuya explicación detallada se dará más adelante. En cualquier caso, la aplicación puede volver a comprobar los permisos del usuario contra su propia base de datos si por seguridad así se prefiere.

A continuación veremos en detalle cómo implementar la estructura del directorio. Para definir los usuarios se utiliza el tipo de objeto *inetOrgPerson* puesto que entre otros incluye atributos para almacenar un identificador de usuario, una contraseña y direcciones de correo electrónico. Para definir las aplicaciones y los permisos se define el nuevo esquema *indect* de las Figura 25 y Figura 26 . Antes de comenzar a explicar el esquema, es importante mencionar varios puntos. El primero es que para definir un nuevo objeto o atributo hay que obtener un OID (*Object Identifier*) de la IANA (*Internet Assigned Numbers Authority*). Un OID es un número que identifica al objeto o atributo de forma única. Por comodidad nosotros utilizaremos los valores reservados para uso experimental por OpenLDAP.org [31][31], 1.3.6.1.4.1.4203.666.XXX, donde X es un número entero. La otra posibilidad sería generar un OID bajo la rama de pruebas 2.25 en http://.oid-info.com/get/2.25. El segundo punto a tener en cuenta es que para definir un nuevo atributo hace falta saber el número que identifica su tipo (*string*, *booleano*, entero,…) y que se denomina *SYNTAX* [32]. En nuestro caso, necesitaremos los tipos:

- *booleano*: su valor de *SYNTAX* es 1.3.6.1.4.1.1466.115.121.1.7.
- *string*: su valor de *SYNTAX* es 1.3.6.1.4.1.1466.115.121.1.15.

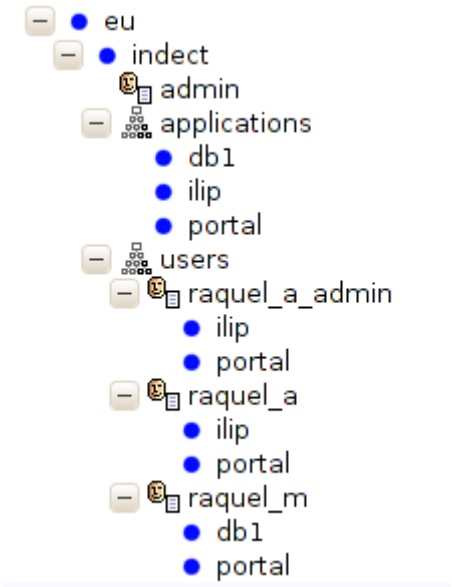


Figura 24. Directorio LDAP para INDECT poblado con aplicaciones, usuarios y permisos.

Una vez tenemos valores para los OIDs y para los campos *SYNTAX* ya podemos implementar el esquema. Las aplicaciones son objetos de tipo *application* que poseen los atributos:

- *aid:* identificador de la aplicación.
- *permissionType:* nombre del tipo de objeto que define los permisos de la aplicación.
- *name:* nombre completo de la aplicación.
- *description:* descripción de la aplicación.
- *mail:* dirección de correo electrónico del administrador de la aplicación.

Los permisos son objetos de tipo base *appPermission*, o de tipos especializados que heredan de *appPermission*, como son de momento *ilipAppPermission* para las aplicaciones web de la plataforma INDECT o *dbAppPermission* para aplicaciones basadas en bases de datos. Cada tipo de permiso posee unos atributos diferentes:

- Un objeto de tipo *appPermission* debe tener como único atributo el identificador de aplicación *aid.*
- *Un objeto de tipo ilipAppPermission* añade un atributo de tipo *string* denominado *admin,* que indica el tipo de usuario.
- Un objeto de tipo *dbAppPermission* añade cuatro atributos de tipo *booleano* denominados *create, read, update* y *delete,* que indican si el usuario tiene dichos permisos o no.

Los atributos *name, description* y *mail* ya existen en el directorio LDAP por lo que sólo fue necesario añadir los demás: *aid, admin, create, read, update, delete* y *permissionType.*

```
# Attribute definition
#string
attributetype ( 1.3.6.1.4.1.4203.666.1
        NAME 'aid'
        DESC 'Application identifier'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)
#string
attributetype ( 1.3.6.1.4.1.4203.666.2
        NAME 'userType'
        DESC 'User type'
        EQUALITY caseIgnoreMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)
#boolean
attributetype ( 1.3.6.1.4.1.4203.666.3
        NAME 'create'
        DESC 'Create permission'
        EQUALITY booleanMatch
        SUBSTR caseIgnoreSubstringsMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
)
#boolean
attributetype ( 1.3.6.1.4.1.4203.666.4
        NAME 'read'
        DESC 'Read permission'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
)
#boolean
attributetype ( 1.3.6.1.4.1.4203.666.5
        NAME 'update'
        DESC 'Update permission'
        EQUALITY booleanMatch
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
)
#boolean
attributetype ( 1.3.6.1.4.1.4203.666.6
        NAME 'delete'
        DESC 'Delete permission'
        EQUALITY booleanMatch
        SYNTAX  1.3.6.1.4.1.1466.115.121.1.7
)
#string
attributetype ( 1.3.6.1.4.1.4203.666.7
        NAME 'permissionType'
        DESC 'Type of permission object'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
)
```

Figura 25. Esquema INDECT para el directorio LDAP (I).

```
# Object Class Definition
objectclass ( 1.3.6.1.4.1.4203.666.8
        NAME 'application'
        DESC 'INDECT application data'
        SUP top
        STRUCTURAL
        MUST ( aid $ permissionType $ name $ mail )
        MAY ( description)
)

objectclass ( 1.3.6.1.4.1.4203.666.9
        NAME 'appPermission'
        DESC 'Especifies that an user (uid) has access to an application (aid)'
        SUP top
        STRUCTURAL
        MUST ( aid )
)

objectclass ( 1.3.6.1.4.1.4203.666.10
        NAME 'ilipAppPermission'
        DESC 'Especifies special permissions in ILIP applications'
        SUP appPermission
        STRUCTURAL
        MUST ( admin )
)

objectclass ( 1.3.6.1.4.1.4203.666.11
        NAME 'dbAppPermission'
        DESC 'Especifies special permissions in DB applications'
        SUP appPermission
        STRUCTURAL
        MUST ( create $ read $ update $ delete )
)
```

Figura 26. Esquema INDECT para el directorio LDAP (II).

En el **Anexo III** se explica en detalle la instalación y configuración de un servidor *OpenLDAP*, junto con la inserción del esquema y la introducción de los datos en el directorio. Aquí veremos la estructura final del directorio. Para organizarlo creamos dos grupos *users* y *applications* de tipo *organizationalUnit*. Bajo *users* se insertan los usuarios y bajo *applications* las aplicaciones. De momento insertamos tres aplicaciones: el portal por el que se accede a las demás aplicaciones (**portal**), un servidor web *ILIP* (**ilip**) y una aplicación de base de datos (**db1**). Bajo la entrada de cada usuario se insertan los permisos que tenga para cada una de ellas, de manera que:

- La ausencia de una entrada de permiso para una aplicación significa que no tiene permisos en la misma.
- Si el permiso es de tipo *appPermission*, significa que tiene permiso de acceso y los permisos por defecto de dicha aplicación.
- Si el permiso es de un tipo específico, tal como *ilipAppPermission* o *dbAppPermission*, además de acceso, el usuario tiene los permisos especificados en él.

El resultado final se observa gráficamente en la anteriormente mencionada Figura 24, y en las Figuras 27 y 28 donde:

- Los tres usuarios **raquel_admin**, **raquel_a** y **raquel_m** tienen permiso de acceso al portal de la plataforma.
- Ambos usuarios **raquel_admin** y **raquel_a** tienen permisos de acceso a la aplicación ilip pero sólo **raquel_admin** es usuario administrador.
- El usuario **raquel_m** tiene permisos de acceso a la aplicación **db1** y también permisos de lectura pero no tiene permisos de creación, ni de actualización, ni de borrado.

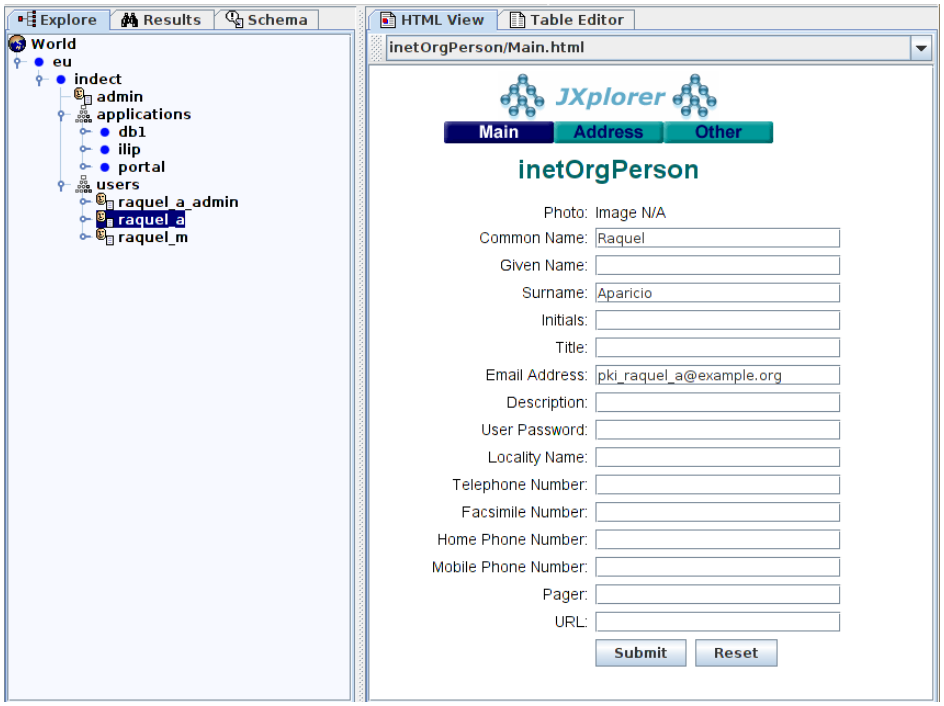


Figura 27. Directorio LDAP: datos del usuario raquel_a.

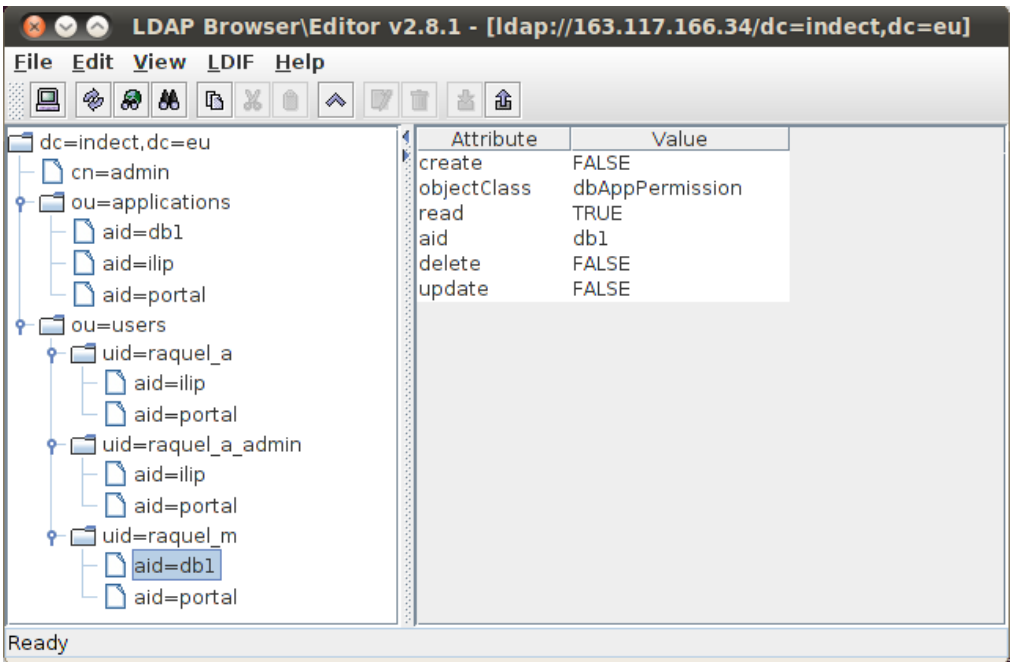


Figura 28. Permisos de raquel_m para la aplicación de base de datos db1.

Una vez dicho servidor está en funcionamiento, hay dos maneras básicas de configurar la interfaz web de la plataforma para utilizarlo:

- A través de Apache: el proceso de autenticación y autorización lo realiza el servidor Apache y es transparente para la plataforma. Si un usuario llega a la plataforma es porque está autorizado para ello.
- A través de la aplicación web de la plataforma: si el usuario no presenta un certificado digital, Apache no hace nada y la aplicación se encarga de pedir los datos de autenticación. Si por el contrario el usuario presenta un certificado, Apache únicamente comprueba su validez: si conoce la CA, si el certificado está revocado, caducado,…. Es la aplicación la que contiene la lógica necesaria para comprobar los datos del usuario. En este caso existe una directiva de Apache que permite decidir qué hacer en estos casos, si ir a la aplicación indicando que el certificado no es válido o directamente devolver un error al usuario.

Para tener mayor control sobre el proceso se ha escogido la segunda opción, realizarlo a través de la aplicación web, lo que requiere modificar los ficheros PHP de la vista y controlador correspondientes a la entrada a la aplicación, es decir, el control de usuarios. Se ha realizado por partes. Dado que la aplicación de por sí ya pedía datos de usuario y contraseña y los contrastaba con los almacenados en su base de datos, la primera modificación ha sido que esta comprobación se realizara contra el servidor LDAP mediante los siguientes pasos básicos:

1. Conectar con el servidor LDAP.
2. Acceder al servidor mediante lo que se denomina un *bind* anónimo (sin credenciales) y obtener el *distinguished name* o DN del usuario a partir de su identificador. Si no se obtiene ningún DN, el usuario no existe y si se obtienen varios DNs indicar al usuario que ha habido un error y que contacte con el administrador.
3. Acceder al sistema (*bind)* con el DN del usuario y la contraseña proporcionada. Si el acceso se realizó con éxito, los datos son correctos. En caso contrario indicar al usuario error de autenticación.
4. Comprobar si el usuario tiene permiso para acceder a la aplicación, es decir, si existe una entrada para la aplicación bajo la entrada del usuario. Si no es así, indicar error de autorización.
5. Comprobar los permisos específicos del usuario dentro de la aplicación, es decir, consultar los valores de la entrada de la aplicación bajo la entrada del usuario.
6. De momento se ha mantenido un nivel extra de seguridad, mediante la comprobación de si el usuario aparece en la base de datos de la aplicación principalmente para obtener datos específicos del mismo, como puede ser la fecha de la última vez en que accedió a la misma o su nombre completo. Si no está en la aplicación indicar error de perfil.

Posteriormente se ha añadido la lógica para manejar los certificados digitales. Es necesario poder acceder a sus campos desde la aplicación web, para lo cual habíamos añadido una directiva al fichero de configuración de Apache de la aplicación web (*/etc/apache2/sites-available/ilip*) en el apartado anterior:

SSLOptions +StdEnvVars

Como se ha explicado, si un usuario presenta un certificado digital durante el establecimiento de la conexión segura, la aplicación puede obtener de él el identificador de usuario y saltarse el paso 3, puesto que la función de autenticación la realiza directamente el propio certificado. En el caso de que, por seguridad, el certificado esté protegido mediante contraseña, es tarea del navegador el solicitarla al usuario y comprobar que es correcta. Si se produce un error con el certificado porque este no sea válido, ya sea porque no contiene el campo del identificador o esté caducado, se procede a realizar una autorización normal con usuario y contraseña indicando el error.

Todo esto se ha conseguido mediante la modificación de la función *login* del controlador de usuarios, de manera que primero compruebe si hay o no certificado y si éste es correcto, para pedir información de autenticación u obtenerla del certificado, además de indicar los posibles errores. Como se ha mencionado, una vez que el usuario se ha autenticado en el sistema, la aplicación comprueba que el usuario existe en la base de datos propia y obtiene datos tales como su nombre o su última fecha de acceso.

3.4 Integración con la herramienta de análisis de relaciones LINK

LINK es un sistema de apoyo para los analistas de la Policía desarrollado por la Universidad de AGH de Ciencia y Tecnología dentro el grupo de trabajo WP3. La versión básica consiste en un conjunto de herramientas para integrar, procesar y visualizar datos de distintas fuentes tales como facturas de teléfono, extractos bancarios o libretas de direcciones. Por ejemplo una de las aplicaciones de dicho sistema es su capacidad de analizar registros de comunicaciones telefónicas entre un grupo de usuarios (llamadas, mensajes de texto SMS y mensajes multimedia MMS) y obtener gráficos que permiten visualizar gráficamente la relación entre los distintos usuarios teniendo en cuenta datos como la frecuencia y duración de las llamadas o el tamaño los mensajes, entre otros. La Figura 29 muestra algunos ejemplos de las distintas maneras de visualizar los datos que ofrece LINK.

Aunque la aplicación original está pensada para comunicaciones telefónicas, debido a que la plataforma decodificadora intercepta comunicaciones a través de Internet, se planteó la posibilidad de utilizarla tanto para llamadas VoIP como para correos electrónicos, de forma equivalente a las llamadas y los mensajes SMS o MMS. Para ello era necesario evaluar el tipo de información que la herramienta necesita como entrada y adaptarlo a los nuevos contenidos. Éstos son los campos que LINK utiliza:

- Fecha de comienzo (campo obligatorio): Fecha y hora en que comenzó la comunicación.
- Fecha de finalización (opcional): Fecha y hora en que finalizó la comunicación.
- Tipo de comunicación telefónica (opcional): Llamada, mensaje de texto SMS o mensaje multimedia MMS.
- Número A llamante (obligatorio).
- Número B llamado (obligatorio).
- IMEI del número A llamante (opcional).
- IMEI del número B llamado (opcional).

- Una serie de campos opcionales con más información acerca del número A llamante o B llamado:
    - BTS CID
    - BTS LAC
    - BTS Name
    - BTS Address
    - BTS Latitude
    - BTS Longitude
    - BTS Range [km]
    - BTS Angle
    - BTS Direction

Se observa que los únicos campos obligatorios son el número llamante, el llamado y la fecha de comienzo de la llamada. La herramienta recibe los datos mediante un fichero *.csv* (*comma separated values*) que es un formato abierto sencillo para representar datos en forma de tabla en un fichero de texto plano. Las columnas se corresponden con los campos de información y están separadas por un carácter determinado, en este caso el punto y coma ";". Cada fila representa una entrada y las filas están separadas por el carácter de nueva línea. La primera fila incluye los nombres de los campos para que la herramienta pueda identificarlos.

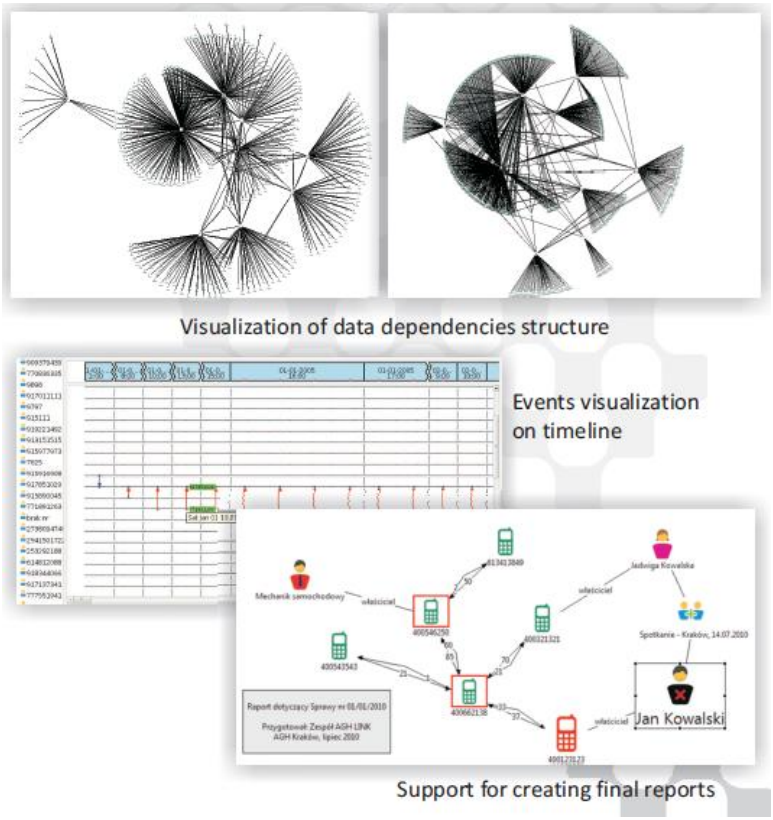


Figura 29. Ejemplos de visualización de datos con la herramienta LINK [30].

Para incluir las llamadas VoIP y los correos electrónicos se han definido dos nuevos

valores para el campo tipo de llamada telefónica: *VOIP* y *EMAIL*. Para llamadas VOIP, la plataforma proporciona la fecha de inicio y la duración, por lo que también es posible calcular la fecha de finalización. Los números llamante y llamado se corresponden con la dirección *sip* de los usuarios. Los emails poseen la fecha en que se enviaron pero obviamente no tienen ni fecha de finalización ni duración. Sin embargo, su tamaño proporciona una información similar, por lo que para este tipo de comunicación se utiliza este valor. Los números llamante y llamado se corresponden con la dirección de correo de los usuarios. En resumen, los campos utilizados son los siguientes:

- Para llamadas VoIP (Figuras 30 y 31):
  - Tipo de comunicación: "*VOIP*".
  - Fecha de comienzo: Fecha de captura.
  - Fecha de finalización: Fecha de captura a la que se suma la duración.
  - Número A llamante: Dirección *sip* origen.
  - Número B llamado: Dirección *sip* destino.
- Para mensajes de correo electrónico (Figura 32):
  - Tipo de comunicación: "*EMAIL*".
  - Fecha de comienzo: Fecha de captura.
  - Tamaño: Tamaño del mensaje en bytes.
  - Número A llamante: Dirección de correo electrónico origen.
  - Número B llamado: Dirección de correo electrónico destino

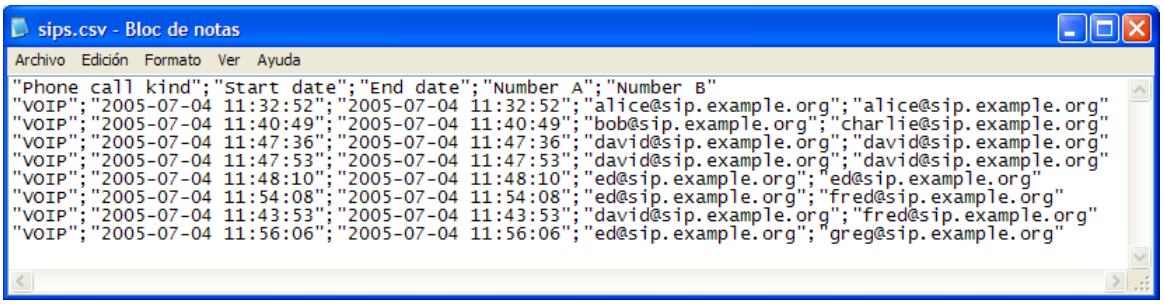


Figura 30. Ejemplo de fichero .csv con datos sobre conversaciones VOIP, visualizado con bloc de notas.

	A	B	C	D	E
1	Phone call kind	Start date	End date	Number A	Number B
2	VOIP	04/07/2005 11:32	04/07/2005 11:32	alice@sip.example.org	alice@sip.example.org
3	VOIP	04/07/2005 11:40	04/07/2005 11:40	bob@sip.example.org	charlie@sip.example.org
4	VOIP	04/07/2005 11:47	04/07/2005 11:47	david@sip.example.org	david@sip.example.org
5	VOIP	04/07/2005 11:47	04/07/2005 11:47	david@sip.example.org	david@sip.example.org
6	VOIP	04/07/2005 11:48	04/07/2005 11:48	ed@sip.example.org	ed@sip.example.org
7	VOIP	04/07/2005 11:54	04/07/2005 11:54	ed@sip.example.org	fred@sip.example.org
8	VOIP	04/07/2005 11:43	04/07/2005 11:43	david@sip.example.org	fred@sip.example.org
9	VOIP	04/07/2005 11:56	04/07/2005 11:56	ed@sip.example.org	greg@sip.example.org

Figura 31. Ejemplo de fichero .csv con datos sobre conversaciones VOIP, visualizado con Excel.

	A	B	C	D	E
1	Phone call kind	Start date	Size	Number A	Number B
2	EMAIL	18/01/2012 16:02	912	alice@example.org	bob@example.org
3	EMAIL	18/01/2012 16:04	860	alice@example.org	bob@example.org
4	EMAIL	18/01/2012 16:09	1967	charlie@example.org	alice@example.org
5	EMAIL	18/01/2012 16:10	2232	charlie@example.org	alice@example.org
6	EMAIL	18/01/2012 16:30	857	charlie@example.org	bob@example.org
7	EMAIL	19/01/2012 13:08	2066	bob@example.org	alice@example.org

Figura 32. Ejemplo de fichero .csv con datos sobre un emails, visualizado con Excel.

El fichero *.csv* se obtiene a través de la interfaz web de la plataforma. Dado que CakePHP utiliza lo que se denomina *layout* para cualquier información que se envía al usuario, ha sido necesario crear uno específico que permita poder enviar los datos en el formato requerido sin que el *layout* general de la aplicación interfiera. En las vistas índice se ha añadido un botón que permite al usuario obtener un fichero *.csv* con los registros que en ese momento está visualizando, es decir, teniendo en cuenta los filtros que hayan sido seleccionados en la aplicación. El botón no es más que una llamada a una función nueva en el controlador correspondiente que se encarga de preparar la exportación a *.csv*: cambia el *layout* al creado, obtiene los campos necesarios de la base de datos para las entradas que cumplan los filtros de contenido activos, y los reenvía a la nueva vista correspondiente. La nueva vista formatea los valores para generar un fichero *.csv* con los campos arriba mencionados: añade delimitadores de texto a los valores individuales y separadores entre los campos, calcula la fecha de finalización a partir de la fecha inicial y la duración de la llamada y procesa las dirección *sip* y de correo para limpiarlas de los caracteres "<" y ">" y de cualquier nombre que la pudiera preceder. Por ejemplo, la dirección "Greg" *<sip:greg@sip.example.org>* se procesaría para obtener simplemente *greg@sip.example.org*.

Debido a que durante el desarrollo de este proyecto la herramienta LINK no estaba directamente accesible para realizar pruebas, lo que se hizo fue enviar los ficheros *.csv* obtenidos en la plataforma a su equipo responsable y ellos a su vez nos proporcionaron los resultados: a partir del fichero *.csv* de prueba mostrado en la Figura 31, LINK generó el grafo de la Figura 33.

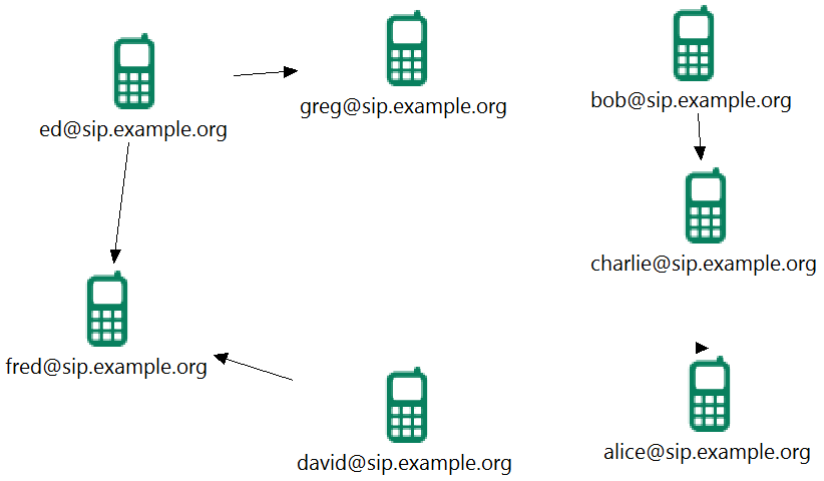


Figura 33. Grafo proporcionado por LINK a partir del fichero *.csv* de la Figura 31.

3.5 Plugins de preclasificación de contenidos

Los *plugins* de pre-clasificación son aplicaciones independientes diseñadas para procesar archivos de un tipo MIME (*Multipurpose Internet Mail Extensions*) determinado. Aunque inicialmente se definieron para el correo electrónico, actualmente los tipos MIME son especificaciones para el intercambio de todo tipo de archivos a través de Internet. La idea es que haya una gran variedad de *plugins*: algunos específicos para analizar tipos muy concretos como por ejemplo archivos PDF (tipo MIME *application/pdf*) o imágenes JPEG (tipo MIME *application/jpeg*) y otros más generales que puedan analizar cualquier tipo de texto (tipo MIME *text/**) o de imagen (tipo MIME *image/**) o incluso cualquier tipo de contenido que definiremos como tipo MIME *. Su tarea es realizar un procesado sobre los contenidos que reciben y devolver dos valores que representan su preclasificación:

- un número del 1 al 5 que indica la relevancia del contenido, 0 en caso de no saber preclasificarlo y -1 en caso de error,
- un mensaje textual que proporciona un poco más de información sobre el procesado y el resultado obtenido.

El motivo de llamar a este proceso preclasificación y no clasificación a secas, es que los valores devueltos son proporcionados por una máquina pero en última instancia debe ser un analista autorizado el que decida su clasificación final. La razón de utilizar el rango de valores de 1 a 5 se debe a que es lo suficientemente amplio pero no tanto como para ser susceptible de interpretaciones personales subjetivas.

Un *plugin* puede ser tanto una aplicación local en la misma máquina en la que se encuentra la estación decodificadora, como una aplicación remota situada en cualquier otra máquina y que debe ser accesible mediante un servicio web, de manera que varias estaciones decodificadoras puedan compartir su uso. El primer tipo es ideal para pequeños *plugins* cuya ejecución no interfiera con el rendimiento de la decodificación de contenidos que realiza *Xplico*, mientras que el segundo parece más adecuado para *plugins* con gran demanda de recursos que puedan ralentizar el sistema o que hagan uso de datos propios que requieran un mantenimiento, como pueden ser listas de contenidos ilegales, de manera que su gestión se simplifica mucho respecto a si hubiera que mantener los datos de cada una de las estaciones decodificadoras.

Los *plugins* ofrecen dos modos de funcionamiento: el modo de *análisis* para preclasificar los contenidos y el modo de *entrenamiento* para modificar el resultado a devolver para un contenido. El caso más sencillo sería un *plugin* que mantiene una lista de contenidos que relaciona cada contenido con el resultado deseado. Esta lista es modificada en el modo entrenamiento y sería consultada en el modo de análisis. Para poder realizar su tarea, estas aplicaciones necesitan recibir el contenido a procesar y cierta metainformación sobre el mismo, que se ha definido a partir de los datos que *Xplico* proporciona tras la decodificación. El *plugin* recibe los parámetros, ejecuta su lógica y devuelve un resultado.

Estas aplicaciones no van a ser invocadas directamente por el analista sino que se va a utilizar un módulo distribuidor intermedio entre *Xplico* y ellas para que puedan ser llamadas de forma automática. Se tratará este módulo en el siguiente apartado. Es necesario almacenar cierta información sobre los *plugins* en la base de datos de la

plataforma, que permita a dicho módulo acceder a ellos y saber cuándo emplearlos. El uso de un módulo distribuidor ofrece varias ventajas importantes frente a la llamada individual:

- Simplifica enormemente la adición o eliminación de *plugins* del sistema.
- Permite realizar el proceso automáticamente en vez de manualmente: el analista encuentra los contenidos ya preclasificados.
- Los contenidos son analizados en paralelo con el consiguiente aprovechamiento de recursos y ahorro de tiempo.
- El módulo se asegura de que cada contenido que reciba es procesado por todos los *plugins* disponibles según su tipo hasta que uno proporcione un resultado o no quede ningún otro *plugin* que pueda procesarlo.

El desarrollo de *plugins* puede realizarse en cualquier lenguaje de programación sin importar que sea remoto o local, ya que para que sean integrables en el sistema basta con que simplemente cumplan los requisitos en la forma de recibir datos y devolver resultados, por lo que siempre es posible realizar un sencillo recubrimiento para lograrlo. Sin embargo, el caso particular de los *plugins* remotos merece especial detalle. Son servicios web accesibles desde la máquina en que se encuentra la estación decodificadora, aunque, para un desarrollador de *plugins*, la comunicación entre el cliente (*Plugin Manager*) y el servidor (servicio web) es transparente si utiliza las herramientas que se le proporcionan. Se han desarrollado algunas plantillas en varios lenguajes de programación como son *bash script* y C y una librería de Java que se explicará bajo el apartado del módulo distribuidor. Estas plantillas básicamente procesan los argumentos de entrada para que el *plugin* pueda acceder a los distintos parámetros individualmente, simplificando así la tarea del desarrollador para que sólo necesite centrarse en el procesamiento del contenido, llegando al punto en que sólo necesita implementar una función de análisis y otra de entrenamiento. La librería es especialmente útil para el desarrollo de *plugins* remotos puesto que abstrae al desarrollador de todo el proceso de entrada y salida mediante mensajes SOAP pero también incluye ciertas utilidades para los *plugins* en general. Ésta librería hace que la implementación de un *plugin* remoto y el mismo *plugin* locales sea prácticamente idéntica, la única diferencia radica en tener en cuenta la descarga del contenido en el caso de los *plugins* remotos, por lo que es muy sencillo implementar un código válido para ambos tipos de *plugins*. En cualquier caso la funcionalidad de descarga de contenidos se proporciona dentro de la librería, mediante el uso de la librería *Apache Commons NET*, que ofrece mayor control sobre las interacciones que la solución adoptada por el desarrollador original de abrir un flujo de lectura directamente sobre la URL del contenido. El **Anexo IV** proporciona varios ejemplos de *plugins* locales y remotos utilizando estas herramientas.

Para definir el formato de entrada de datos a los *plugins* se tuvo en cuenta que los datos que recibe son proporcionados por el módulo distribuidor de contenidos, que a su vez los recibe bien de *Xplico* o mediante algún tipo de procesamiento, como se verá en seguida. La información concreta que *Xplico* proporciona depende del tipo de contenido que se procese, por lo que se decidió partir de elementos comunes a todos los tipos:

- Categoría del caso.
- Identificador del caso: se utiliza el identificador de la base de datos aunque

también podría ser el nombre.

- Ruta del contenido: ruta completa al contenido. *Xplico* proporciona una ruta dentro del sistema local y el módulo distribuidor se encargada de pasarla tal cual para *plugins* locales y transformarla en una URL, utilizando la dirección IP del servidor, para *plugins* remotos.
- Nombre del contenido: nombre del archivo a procesar.
- Marca de tiempo de la captura: indica cuándo se realizó la captura del contenido; aporta información que puede ser importante.

Además es interesante que el *plugin* reciba algo de información extra. Los siguientes campos de la llamada al *plugin* son obtenidos por el módulo distribuidor:

- Tipo MIME del contenido.
- Tamaño del contenido.
- *Hash* del contenido.

Como resumen, la llamada a un *plugin* local se realiza siguiendo el formato:

ruta_al_plugin/ejecutable analysis *valor_case_category*
valor_case_identifier
valor_content_uri
valor_content_name
valor_content_timestamp
valor_content_type
valor_content_hash
valor_content_size

ruta_al_plugin/ejecutable training *valor_case_category*
valor_case_identifier
valor_content_uri
valor_content_name
valor_content_timestamp
valor_content_type
value_ontent_hash
valor_content_size
valor_process_relevance
valor_process_message

El *plugin* debe proporcionar su salida imprimiendo por salida estándar los comentarios y devolviendo la relevancia como resultado de la ejecución. La comunicación con un *plugin* remoto se realiza mediante los siguientes mensajes SOAP, generados por las capas de módulo distribuidor situadas en ambos extremos:

Petición de análisis:

<analysisRequest>

<caseCategory>*valor_case_category* </caseCategory>

<caseID>*valor_case_id* </caseID>

<contentURI>*valor_content_URI* </contentURI>

<contentName>*valor_content_name* </contentName>

<contentTimestamp>*valor_content_timestamp* </contentTimestamp>

<contentType>*valor_content_type* </contentType>

<contentHash>*valor_content_hash* </contentHash>

<contentSize>*valor_content_size* </contentSize>

</analysisRequest>

Respuesta de análisis:

<responseAnalyze>

<contentURI>*valor_content_URI* </contentURI>

<contentName>*valor_content_name* </contentName>

<contentHash>*valor_content_hash* </contentHash>

<contentSize>*valor_content_size* </contentSize>

<processTimestamp>*valor_process_timestamp* </processTimestamp>

<processRelevance>*valor_process_relevance* </contentRelevance>

<processMessage>*valor_process_message* </processMessage>

</responseAnalyze>

Petición de entrenamiento:

<trainingRequest>

<caseCategory>*valor_case_category* </caseCategory>

<caseID>*valor_case_id* </caseID>

<contentURI>*valor_content_URI* </contentURI>

<contentName>*valor_content_name* </contentName>

<contentTimestamp>*valor_content_timestamp* </contentTimestamp>

<contentType>*valor_content_type* </contentType>

<contentHash>*valor_content_hash* </contentHash>

<contentSize>*valor_content_size* </contentSize>

<processRelevance>*valor_process_relevance* </processRelevance>

<processMessage>*valor_process_message* </processMessage>

</trainingRequest>

Respuesta de entrenamiento:

<responseTrain>

<contentURI>*valor_content_URI* </contentURI>

<contentName>*valor_content_name* </contentName>

<contentHash>*valor_content_hash* </contentHash>

<contentSize>*valor_content_size* </contentSize>

<processTimestamp>*valor_process_timestamp* </processTimestamp>

<processRelevance>*valor_process_relevance* </contentRelevance>

<processMessage>*valor_process_message* </processMessage>

</responseTrain>

3.6 Módulo distribuidor de contenidos

El distribuidor de contenidos es un módulo software, creado para recibir los contenidos decodificados por *Xplico* y distribuirlos a los *plugins* preclasificadores que se encargan de su procesado y etiquetado; por ello también recibe el nombre de *Plugin Manager*. Ambos fueron diseñados conjuntamente en un proyecto fin de carrera anterior [6] pero han sido actualizados y adaptados a lo largo de la integración, llegando a rehacer gran parte del código. Como se ha explicado ya, los *plugins* son pequeñas aplicaciones que reciben un contenido y su metainformación, lo procesan y devuelven una preclasificación compuesta de un número que indica su relevancia y un comentario que ofrece algo más de información sobre el procesamiento realizado. Estos *plugins* se especializan en un tipo MIME (*Multipurpose Internet Mail Extensions*) determinado. Una vez configurados en el sistema mediante la asociación del *plugin* con dicho un MIME, cada vez que el *Plugin Manager* recibe un contenido, comprueba su tipo e invoca los *plugins* capaces de procesarlo. Aparte de analizar contenidos, los *plugins* también pueden ser entrenados para mejorar la preclasificación de futuros contenidos.

En su primera versión, el *Plugin Manager* no había sido integrado con *Xplico* sino que se había probado mediante una interfaz web de entrada que permitía su uso como una aplicación independiente, pasándole como argumentos los datos que *Xplico* le habría de enviar. Esto implica que el desarrollador inicial carecía de los conocimientos sobre la arquitectura interna de *Xplico* que sólo ha sido posible adquirir durante el desarrollo de este proyecto y que han llevado a la modificación del código original. En este apartado, se explicarán los pasos realizados para integrar ambas herramientas, que inicialmente dividen en tres tareas fundamentales:

- Combinar las bases de datos de ambas herramientas en una única.
- Combinar la interfaz web de ambas herramientas.
- Realizar la llamada al *Plugin Manager* directamente dentro del código de *Xplico*, de manera que el funcionamiento sea automático.

Los datos que el *Plugin Manager* requiere para realizar el análisis de contenidos, es decir, información acerca de los *plugins* y las reglas para ejecutarlos, se almacenaba originalmente en una base de datos propia, de tipo SQLite para facilitar la posterior integración con la base de datos de *Xplico*. El registro de *plugins* almacena información sobre su nombre, ubicación o URI, tipo MIME de los contenidos que puede procesar y una breve descripción del *plugin*. A la hora de invocar el *plugin* se hace uso del campo de ubicación por lo que en el caso de *plugins* locales éste debe contener la ruta al ejecutable que lo implementa, y en el caso de los *plugins* remotos, la URI del servicio web para acceder al mismo. El registro de reglas contiene entradas que asocian tipos de contenido con *plugins* y el orden en que deben ejecutarse. Los tipos MIME incluídos en las reglas pueden ser más o menos específicos, de manera que *plugins* que sean capaces de procesar cualquier tipo de contenido pueden ser asociados a una regla con un tipo * mientras que *plugins* diseñados para procesar un tipo MIME muy concreto sean asociados única y exclusivamente al mismo. En caso de que existan múltiples *plugins* para analizar el mismo tipo de contenido, el orden de ejecución permite que se apliquen primero los más específicos y, si no son capaces de obtener una preclasificación, se proceda a ejecutar los más generales. Básicamente, cuando el *Plugin Manager* recibe un contenido de *Xplico* para ser procesado, comprueba su tipo y obtiene una lista ordenada de los *plugins* que lo pueden analizar según las reglas que haya configuradas en ese

momento. El contenido va siendo analizado *plugin* a *plugin* en el orden configurado hasta que uno de ellos sea capaz de devolver una relevancia o no haya más *plugins* disponibles para ese tipo de contenido. El resultado final de la preclasificación es devuelto a *Xplico*, que lo almacena actualizando la entrada del contenido en la base de datos. La base de datos también almacena una caché de contenidos analizados, lo que permite aligerar el procesamiento de contenidos que hayan sido procesados previamente puesto que ya no es necesario ejecutar los *plugins*. Para ello se almacenan el *hash* y el tamaño del contenido para identificarlo y las respuestas que se obtuvieron tras su procesamiento. La integración de las bases de datos consistió en insertar las tablas que almacenan toda esta información en la base de datos común.

Por último, el *Plugin Manager* incluye una interfaz web que permite la fácil inserción y eliminación de *plugins* del sistema así como la creación y borrado de las reglas que asocian los tipos de contenidos con los *plugins* que los deben analizar e indican en qué orden aplicarlos.

Veamos ahora el diseño original de comunicación entre las distintas partes: *Xplico*-módulo distribuidor y módulo distribuidor-*plugins*. Los *plugins* necesitan cierta metainformación acerca del contenido que van a procesar, pero el *Plugin* M*anager* es un intermediario para el que sólo el tipo de contenido es necesario. Dado que gran parte de la información obtenida por *Xplico* depende precisamente del tipo de contenido procesado, el desarrollador original del *Plugin Manager* decidió establecer una interfaz en la que se definen elementos comunes a todos los tipos de contenido pero que permite enviar más datos mediante una serie de campos sin formato específico. Además de los datos que precisan los *plugins*, la interfaz de comunicación *Xplico-Plugin Manager* incluye varios campos más, de manera que la lista completa es:

- Categoría del caso: por ejemplo tráfico de drogas, armas, pedofilia.... Ofrece información sobre el contexto en el que se está investigando.
- Identificador del caso: es el identificador de la base de datos, puede utilizarse por ejemplo para relacionar contenidos entre sí.
- Ruta del contenido: ruta completa dentro del sistema en que se encuentra el archivo que almacena el contenido. El módulo distribuidor se encarga de convertirla en una URI para que los *plugins* remotos puedan descargar el contenido mediante FTP, como se verá más adelante.
- Nombre del contenido: nombre del archivo a procesar, que se puede obtener de la ruta completa por lo que puede estar vacío.
- Tipo de contenido: si está vacío el módulo se encarga de obtenerlo a partir del archivo, ya que es necesario para decidir qué *plugins* analizarán el contenido.
- Marca de tiempo de la captura: indica cuándo se realizó la captura del contenido; no es obligatorio pero aporta información que puede ser importante.
- Contenidos relacionados: permite enviar la ruta de contenidos relacionados con el principal, como pueden ser los adjuntos de un correo, o que el *plugin* devuelva la ruta a contenidos que considera relacionados con el que acaba de procesar. Puede estar vacío.
- Atributos sobre el contenido: cualquier otra información acerca del contenido. Puede estar vacío.

- Datos de contexto: este campo será explicado más adelante.

Para el modo de entrenamiento de *plugins*, la entrada del *Plugin Manager* recibe otros dos campos que indican la relevancia y el mensaje a devolver para ese contenido.

Los *plugins* reciben y devuelven datos de una manera muy concreta: proporcionan como código de salida la preclasificación numérica e imprimen por la salida estándar una descripción textual de la preclasificación. Una vez el *Plugin Manager* ha obtenido ambos valores, se comunica de nuevo con *Xplico* utilizando un mensaje con el formato siguiente:

1. URI del contenido.
2. Nombre del contenido.
3. *Hash* del contenido.
4. Tamaño del contenido.
5. Marca de tiempo del procesamiento.
6. Relevancia del contenido.
7. Comentarios generados durante el procesamiento.
8. Contenidos relacionados.
9. Atributos del contenido.
10. Datos de contexto: se corresponden con los recibidos en la petición.

Veamos ahora un par de detalles importantes que surgieron durante la integración. Partimos de la base de que *Xplico* envía una petición de análisis por cada contenido que decodifique y almacene en la base de datos, y debe recibir la respuesta con los resultados para actualizar la información de la base de datos. Para evitar que *Xplico* se quedara bloqueado durante el procesamiento de cada contenido y pudiera continuar procesando la captura, el *Plugin Manager* se desarrolló para funcionar de forma asíncrona: cada vez que *Xplico* decodifica un contenido, lanza una petición y continúa con la captura, mientras que el *Plugin Manager* se encarga de controlar el procesamiento de cada uno de los contenidos y contacta de nuevo con *Xplico* cada vez que alguno termine de ser procesado. Esto tiene varias implicaciones:

- el *Plugin Manager* necesita saber cómo contactar de vuelta con *Xplico*,
- *Xplico* necesita incluir en la petición cierta información de contexto que le permita identificar el contenido dentro de su base de datos, para que le sea devuelta junto con el resultado,
- es necesario un control que evite que la aplicación finalice antes de que se hayan terminado de procesar todos los contenidos.

La solución adoptada por el desarrollador original consistió en añadir en el código de *Xplico* una función denominada de *callback* o retorno, cuyo puntero se pasa al *Plugin Manager* durante la inicialización del sistema, y a través de la que dicho módulo contacta de nuevo con *Xplico*, y enviar en la petición ciertos datos de contexto que permitan a *Xplico* identificar el contenido, y que son devueltos en dicha función de *callback*. Sin embargo, al no llegar a realizar la integración, no se realizó ningún paso de cara a controlar la finalización del proceso.

Originalmente se definió el siguiente formato de llamada de *Xplico* al *Plugin Manager*:

1. Categoría del caso.
2. Identificador del caso.
3. Ruta al contenido.
4. Nombre del contenido.
5. Tipo del contenido.
6. Otros datos.

El último elemento se mantuvo sin definir. Durante la integración se comprobó que lo más práctico era que almacenara información para acceder directamente al contenido dentro de la base de datos, básicamente:

- Nombre de la tabla en que se encuentra e identificador dentro de la misma.
- Si depende de otro contenido que llamaremos padre, nombre de su tabla e identificador.
- Datos del caso y sesión.

Como ya se ha mencionado, fue necesario realizar actualizaciones en el código para poder realizar la integración satisfactoriamente. Se mencionarán únicamente las modificaciones más importantes que han afectado al diseño o funcionamiento del *Plugin Manager* original. Por ejemplo, se había desarrollado lo que se denominaba una capa de aplicación que permitía invocar el *Plugin Manager* como un programa independiente, bien para un contenido que se pasaba por argumento o para una lista de contenidos presentes en una base de datos configurada de antemano, indicando el modo de funcionamiento. El propósito de dicha capa de aplicación era únicamente demostrar el funcionamiento del módulo e incluso usaba algunos valores por defecto para algunos de los campos necesarios por lo que no era directamente integrable en el sistema. Las otras capas del módulo son:

- Capa lógica: se encarga de obtener la información sobre los *plugins* que pueden procesar el contenido, preparar los datos a pasar a la capa de transporte y recibir la respuesta de la capa de transporte local o remoto y enviarla de vuelta a *Xplico*.
- Capa de transporte: se encarga de distinguir entre *plugins* remotos o locales y enviar la petición a la capa correspondiente.
- Capa de transporte local: se encarga de preparar y ejecutar la llamada a un *plugin* local y enviar la respuesta a la capa lógica.
- Capa de transporte remoto: se encarga de preparar y ejecutar la llamada a un *plugin* remoto y enviar la respuesta a la capa lógica.

Esta es la estructura de la denominada parte cliente del módulo, denominada así porque cumple las funciones del cliente en un modelo cliente-servidor que es el que se utiliza para ejecutar *plugins* remotos. Éstos están situados en una máquina remota a la que se accede mediante un servicio web. Cada *plugin* remoto se corresponde con un servicio web diferente y, dado que la entrada al servicio consiste en tareas comunes como es procesar la petición SOAP para obtener los datos y enviarlos al *plugin*

propiamente dicho, se programó un código de entrada al servicio común denominado parte servidora del *Plugin Manager* y que se incluía en todos los *plugins* remotos. Esta parte servidora se programó en Java y se divide a su vez en la parte invariable que constituye la entrada al servicio, y en una interfaz y una clase que la implementa donde se debe programar el *plugin* o, en caso de que el *plugin* se desarrolle en cualquier otro lenguaje, ejecutar una llamada al mismo.

En cuanto a la parte cliente, la solución adoptada para su integración fue eliminar la capa de aplicación y compilar el módulo como una librería para que *Xplico* la importe de forma estática a través de su módulo distribuidor *lite*, de forma que pueda accederse directamente a las funciones del *Plugin Manager* pasándole los datos necesarios. Se ha elegido el módulo distribuidor *lite* porque se considera el mejor punto en *Xplico* para realizar dichas llamadas, ya que tras la decodificación de los contenidos y la inserción de su información en la base de datos, prácticamente todos los datos acerca del contenido están disponibles: identificador del caso, ruta del contenido, nombre del contenido y marca de tiempo de la captura, y los demás bien se obtienen de la base de datos (categoría) o bien pueden estar vacíos, como es el caso del tipo de contenido, que el *Plugin Manager* se encarga de obtener si no le ha sido proporcionado, o los contenidos relacionados y los atributos del contenido, que son opcionales.

Debido a que el *Plugin Manager* había sido desarrollado en C++ y *Xplico* en C, el primero paso fue crear una interfaz que permitiera la intercomunicación, incluyendo funciones:

- Que creen y gestionen la instancia de la clase del *Plugin Manager* que va a distribuir los contenidos. C no entiende de clases y objetos por lo que el objeto *Plugin Manager* creado debe ser mantenido por *Xplico* y pasado como argumento en cada petición de manera que la interfaz realice la llamada al método correspondiente de la clase.
- Que permitan que *Xplico* realice llamadas con datos de tipo C, convirtiéndolos a los tipos de C++ para invocar las funciones correspondientes del *Plugin Manager.*
- Que permitan al *Plugin Manager* invocar la función de *callback* indicada por *Xplico* con tipos de datos de C.

Con este paso realizado, se observaron problemas a la hora de procesar los contenidos asíncronamente y se descubrió que tenía relación con la manera de funcionar de *Xplico* y del *Plugin Manager*. En el apartado de *Estado del Arte: Xplico* se explicó su funcionamiento a grandes rasgos. En este punto es necesario entrar un poco más en detalle. Cuando el sistema arranca, el único módulo que se inicializa es el DEMA, que queda corriendo como un demonio. Hasta que no hay capturas presentes, el DEMA no lanza el módulo *Xplico* y otros cuatro módulos: manipuladores para chat de *facebook*, *webmail* y contenidos HTTP y un agregador para *paltalk express*. Todos estos módulos son lanzados por el DEMA en modo multi-hilo, mediante una piscina de 100 hilos, de forma que se van lanzando hilos según se van encontrando datos que analizar en la captura. Una vez que los datos han sido procesados y los contenidos obtenidos y almacenados, los hilos van terminando su ejecución por lo que se renueva el espacio para lanzar nuevos hilos. El módulo *Xplico* es el encargado de comprobar si todos los hilos han terminado su ejecución antes de terminar la suya propia y dejar solo el demonio del DEMA hasta que este reconozca nuevas capturas y reinicie el

procedimiento. Sin embargo, el funcionamiento del *Plugin Manager* asumía que *Xplico* funcionaba como un único proceso que iba procesando los datos de la captura secuencialmente y mandándolos a los módulos que fuera preciso. De esta manera, el *Plugin Manager* se diseñó como una clase de la que se crearía una única instancia y que iría lanzando hilos para procesar asíncronamente los contenidos que recibiera. El problema surgió de que originalmente no se realizaba ningún tipo de control sobre la creación de los hilos. Durante las pruebas de integración se observó que a veces había muchos contenidos sin procesar y se descubrió que el motivo era que llegado cierto punto los hilos que el *Plugin Manager* intentaba lanzar no eran creados por falta de memoria, debido a que *Xplico* ya de por sí lanzaba esos 100 hilos. Por ello se decidió implementar una piscina de otros 100 hilos en el *Plugin Manager*, de manera que cuando todos los hilos están siendo utilizados los nuevos contenidos han de esperar a que algún hilo quede libre. Se modificó tanto el código de creación de hilos añadiendo un contador para asegurar que no se lanzan más hilos de los permitidos como el código de terminación de *Xplico* para que espere a que los hilos activos terminen su procesamiento una vez que la captura haya sido totalmente decodificada. Esto se realizó mediante mecanismos de exclusión mutua (*mutex*), de acceso de variables y de señalización del cumplimiento de ciertas condiciones por parte de ciertas variables para evitar el uso de esperas activas.

Sin embargo, posteriormente aparecieron otra serie de errores debido a la compleja sincronización entre los hilos del *Plugin Manager* y de *Xplico*, por lo que se decidió directamente rehacer todo el código de la denominada parte cliente del *Plugin Manager*. Además se ha aprovechado para eliminar la interfaz de traducción de tipos de datos, permitiendo el manejo directo de datos de tipo C. Dado que el propio *Xplico* hace uso de una piscina de hilos se ha decidido implementar una interfaz síncrona que ejecuta todo el proceso de preclasificación en el mismo hilo que ejecuta la llamada. En cuanto a la base de datos, en lugar de abrir una base de datos dedicada en una ruta fija codificada directamente en el módulo, éste recibe el objeto de la base de datos junto con el objeto *mutex* que regula su acceso, para evitar los errores debidos a intentos de escritura sobre la base de datos cuando esta está siendo utilizada por otro hilo, errores que se producían habitualmente en el código original debido a la falta de este control y que se solucionaban reintentando la escritura. Por lo demás el diseño en capas del *Plugin Manager* original se ha mantenido, simplemente mejorando la estructura e implementación del código, como por ejemplo:

- Se ha reducido el código reaprovechando una misma estructura de datos tanto para el caso de análisis como el de entrenamiento puesto que el procesado de la misma a lo largo de las capas del módulo del *Plugin Manager* es prácticamente el mismo y el uso de dos estructuras simplemente duplicaba y complicaba el código,
- Se permite indicar al *Plugin Manager* qué *plugin* debe procesar el contenido, funcionalidad que se utiliza para realizar un entrenamiento desde la interfaz web.
- Se obtiene la marca de tiempo del procesado de contenidos, que antes no se realizaba,
- No se actualiza la cache cuando un contenido no ha podido ser preclasificado por ninguno de los *plugins* disponibles,
- Se realiza un control de errores más exhaustivo.

Además, se ha creado una aplicación que hace uso de esta librería y que permite ejecutar el módulo distribuidor de forma separada de *Xplico*. Como se ha mencionado esto es útil para el entrenamiento de los *plugins*, que se realiza desde la web de forma individual, seleccionando el contenido y la relevancia y mensaje que se desean obtener.

En cuanto a la parte servidora para los *plugins* remotos, se realizaron primero pequeñas adaptaciones hasta proceder a un rediseño del funcionamiento, básicamente mediante la separación total de la parte receptora de la petición, lo que a partir de ahora será la parte servidora, respecto al procesamiento que realmente corresponde al *plugin*. Veamos los cambios en detalle: la programación de *plugins* remotos tiene una complicación adicional al desarrollarse como servicios web, con los que la comunicación se realiza mediante SOAP. Originalmente, el proyectando que diseñó el *Plugin Manager* y los *plugins* decidió que las peticiones SOAP procedentes del *Plugin Manager* fueran recibidas por un módulo Axis dentro de un servidor Tomcat, para lo que desarrolló un paquete en Java, considerando que el *Plugin Manager* estaba dividido en dos partes: el lado cliente que enviaba las peticiones (el integrado con *Xplico*, desarrollado en C) y el lado servidor que recibía las peticiones (el paquete Java proporcionado). Sin embargo, el lado servidor no consistía en un módulo genérico que procesara peticiones e invocara *plugins* situados en su máquina, sino que debía ser replicado para cada *plugin*, modificando ciertos ficheros de configuración y una clase concreta donde se insertaría la funcionalidad del *plugin*, con la idea de que fuera programada directamente en dicha clase, aunque en realidad también era posible:

- cargar otra clase Java que contuviera dicha funcionalidad.
- realizar una llamada a un comando externo, lo cual sirve para *plugins* desarrollados en otro lenguaje diferente de Java.

En cualquiera de los dos casos, seguía siendo necesario modificar la clase original del paquete, indicando los datos de la clase a cargar o el comando a ejecutar. El resultado final era un servicio web por *plugin* remoto, mediante la replicación innecesaria de mucho código completamente idéntico - el del procesamiento de los datos de la petición - y el inconveniente de que los desarrolladores tuvieran que modificar ficheros dentro del paquete proporcionado.

En este proyecto se ha mejorado este paquete, mediante la modificación de las clases existentes y la creación de otras nuevas, para facilitar la tarea del desarrollador Java y minimizar la replicación de código común. La funcionalidad se mantiene separada completamente de manera que:

- El código correspondiente a la ejecución de *plugins* locales se almacena en un lugar conocido por el sistema.
- El código correspondiente a la ejecución de *plugins* remotos se inserta en la librería del servicio Axis. Se encarga de la entrada y salida de información de los *plugins* remotos.
- El código necesario para el desarrollo de *plugins* se proporciona a los programadores en forma de una estructura de ficheros que incluye un script de compilación basado en unas convenciones, y algunas clases con utilidades.

El programador simplemente necesita seguir una sencilla convención a la hora de generar los ficheros que contienen el *plugin* local y remoto para asegurarse de que el sistema es capaz de ejecutar sus *plugins*. Es necesario comentar que en caso de no querer utilizar dicho paquete (tanto el original como el mejorado), el *plugin* remoto debe incluir todo el procesamiento de la petición SOAP y el procesamiento de la caché remota. El desarrollo de un *plugin* en JAVA sencillo, donde el mismo código es válido tanto para remoto como para local, para entender cómo utilizar la librería se presenta en el **Anexo IV.**

Aparte de utilizar el paquete para el desarrollo de *plugins* y configurar la plataforma ILIP con los datos del nuevo *plugin*, también es necesario realizar una pequeña configuración del sistema en que está se encuentra, puesto que, por seguridad, se ha tomado la decisión de que cada *plugin* descargue el contenido a analizar mediante FTP a través de su propio usuario virtual. Expliquemos esto con un poco más de detalle. Los *plugins* locales tienen acceso local a los contenidos que deben analizar, pero no es así para los *plugins* remotos que deben descargarse el contenido antes de poder proceder a analizarlo. Es por ello que el *Plugin Manager* modificaba el campo ruta al contenido en una petición a un *plugin* remoto, convirtiendo una ruta local en una URL para que el *plugin* pudiera descargar el contenido. Para ello, es necesario primero instalar un servidor de contenidos en la máquina en que se sitúa la plataforma: VSFTP. En el código original, la URI generada por el *Plugin Manager* incluía ya toda la información necesaria para la descarga, tanto la IP de la máquina como los datos (usuario y contraseña) del usuario local, de manera que los *plugins* remotos sólo tenían que conectarse directamente a la URL y guardar el fichero con el contenido. Esto se ha mejorado de manera que cada *plugin* remoto posee su propio usuario virtual para descargar contenidos. La URI enviada por el *Plugin Manager* únicamente incluye la dirección IP y la ruta relativa del contenido bajo el directorio en que por seguridad se encierra a los usuarios FTP y el *plugin* debe conectarse al servidor VSFTP, entrar mediante su propio usuario y contraseña y solicitar la descarga del contenido. En el **Anexo V** se explica en detalle todo el proceso para instalar un servidor de *plugins* utilizando el servidor web *Apache Tomcat* con el módulo *Axis*.

3.7 Módulo de Transcripción VoIP-Texto

Este módulo incorpora el motor de reconocimiento de voz Sphinx-4 [33] que transcribe automáticamente las conversaciones VoIP (*Voice Over Internet Protocol*) interceptadas de audio en formato *.wav* o *.mp3* a texto. El analista pasa de tener que transcribir él mismo las conversaciones a simplemente comprobar si son correctas y corregirlas en caso contrario.

Un sistema de reconocimiento automático del habla, cuyos bloques básicos se muestran en la Figura 34, hacen uso de los siguientes materiales:

- El *speech corpus* es una base de datos que contiene ficheros de audio con grabaciones de voces de individuos y sus correspondientes ficheros de texto.
- Un modelo acústico creado a partir del *speech corpus* de un determinado individuo y que será utilizado por los motores de reconocimiento de voz. Consiste en representaciones estadísticas de los sonidos que componen cada palabra. La obtención de este modelo se corresponde con el entrenamiento.

- Un modelo de lenguaje consistente en un fichero con secuencias de palabras relacionadas con sus distintas probabilidades de aparición. Se utiliza para intentar predecir la siguiente palabra en una conversación.
- Un modelo de gramática es el equivalente a un modelo de lenguaje pero para aplicaciones dedicadas al reconocimiento de un número limitado de palabras. Básicamente, se reduce el vocabulario pero se agiliza el proceso.
- Un diccionario provee las reglas de pronunciación para las palabras del modelo de lenguaje.

En cuanto al caso particular de Sphinx-4 cabe destacar que se trata de código Open Source desarrollado por el grupo CMU Sphinx en colaboración con la Universidad de Carnegie Mellon, los laboratorios Sun Microsystems y Mitsubishi Electric y Hewlett Packard, con contribuciones de la Universidad de California en Santa Cruz y del Massachusetts Institute of Technology (MIT). Esta versión está totalmente desarrollada en Java, muy documentada y dispone de distintas capacidades de reconocimiento en cuanto al tamaño de vocabulario. El modelo de lenguaje con el mayor vocabulario disponible en Sphinx-4 se denomina HUB4, contiene 64000 palabras en inglés y al aplicarlo se obtiene una tasa de error teórica de un 20%. El diccionario más extenso disponible en Sphinx-4 es el *cmudict.0.6.d*. Además Sphinx-4 es el sistema con soporte para el mayor número de idiomas y es el único que dispone de modelos acústicos entrenados para conversaciones telefónicas.

Este módulo de transcripción de llamadas VoIP fue diseñado en forma de scripts que se encargan de realizar las distintas tareas necesarias: transcripción, entrenamiento y preparación previa de la estructura de ficheros necesaria. Para realizar la transcripción el sistema además hace uso de un fichero de configuración XML en el que hay que indicar dónde se encuentran varios componentes tales como el modelo acústico y el modelo de lenguaje.

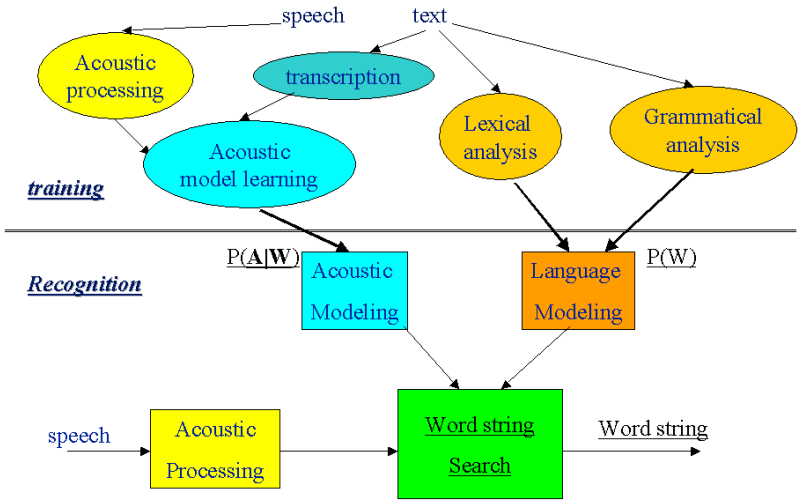


Figura 34. Bloques básicos de un sistema de reconocimiento del habla [34].

La funcionalidad de entrenamiento se basa en SphinxTrain y permite mejorar la calidad de las transcripciones de un usuario mejorando el modelo acústico del mismo a partir del *speech corpus* que se tenga, por ejemplo de conversaciones que se hayan

interceptado previamente. Es necesario mencionar que lo que el desarrollador original denomina entrenamiento en realidad es más bien una adaptación del modelo genérico HUB4 al usuario, mediante los audios y las transcripciones disponibles. En cualquier caso hay que tener en cuenta que para que el modelo acústico sea bueno, el *speech corpus* debe serlo también, lo que se traduce en que hay que asegurarse de que las conversaciones que se utilicen estén bien transcritas.

Al igual que en el caso del *Plugin Manager,* el desarrollador del módulo decidió almacenar la información en una base de datos de tipo SQLite, con vistas a una fácil integración con *Xplico*. El sistema también incluye una interfaz web desarrollada en CakePHP a través de la que se puede acceder a la conversación ya transcrita dividida mediante marcas de tiempo en la que se indica el interlocutor que habla en cada momento. Tanto la conversación como las marcas de tiempo pueden ser modificadas para mejorar la transcripción y realizar el entrenamiento para obtener un modelo acústico. Más adelante se tratarán los cambios en la base de datos y en la interfaz web pero veamos primero cómo se ha realizado la integración en *Xplico* y los cambios que se han realizado en los *scripts*.

El módulo incluye los siguientes scripts:

- *audiosplit.sh*: genera la estructura de ficheros necesaria para el entrenamiento y se encarga de dividir la conversación en ficheros de audio individuales consistentes en frases divididas mediante las marcas de tiempo presentes en la transcripción. Recibe como argumentos los ficheros de audio *.wav* de los hablantes, la conversación transcrita y el *padding* o margen temporal a añadir sobre las marcas de tiempo obtenidas de la transcripción (permite solucionar la limitación de precisión de Sphinx-4 a la hora de generar las marcas de tiempo, que se sitúa en unos 500 milisegundos).
- *files.sh*: crea ciertos archivos necesarios para el entrenamiento. Recibe como argumentos los ficheros de audio *.wav* de los hablantes y la conversación transcrita.
- *timestamps.sh*: modifica la marca de tiempo de un determinado fragmento de audio y actualiza el correspondiente fichero de audio. Recibe como argumentos el fichero de audio *.wav* del usuario, el fragmento a modificar, la conversación transcrita y el valor antiguo y nuevo de la marca de tiempo a modificar, que no deben coincidir con ninguna otra marca.
- *training.sh*: realiza el entrenamiento para un determinado usuario. Recibe como argumento el fichero de audio del usuario.
- *transcript.sh*: realiza la transcripción de una conversación y almacena los datos necesarios en la base de datos. Recibe como argumentos los ficheros de audio .mp3 individuales de los hablantes, que convierte a *.wav* con cambio de frecuencia a 16KHz para poder procesarlos, y el nombre a dar a la conversación.

A parte de integrar las bases de datos y la interfaz web, este planteamiento de scripts parece indicar que para la transcripción automática bastaría con invocar *transcript.sh* desde *Xplico* cuando el contenido que se haya decodificado corresponda a un fichero de audio y, para el entrenamiento, bastaría llamar a los demás scripts en el orden oportuno desde la interfaz. Sin embargo, esto resultó no ser tan directo puesto

que algunas de las decisiones tomadas por el desarrollador original no resultaron prácticas a la hora de hacer su integración. Fue necesario realizar profundas modificaciones a todos los niveles: scripts, base de datos e interfaz web. La mayoría de los cambios están relacionados entre sí: cambios en la base de datos tanto por simplicidad como por comodidad o necesidad implicaron cambios en los scripts y en la interfaz web, y relacionar los audios con las transcripciones sólo era posible añadiendo campos adicionales en la base de datos.

Lo primero que se realizó fueron los cambios en los scripts y la unificación de las dos bases de datos, sin hacer cambios en la estructura original. En ese punto se observaron posibles mejoras de la base de datos pero se prefirió integrar primero la interfaz web y realizar las mejoras al final, lo que implicaba retocar de nuevo la interfaz web, pero permitía minimizar posibles errores.

El formato inicial de la base de datos del módulo de transcripción consistía en dos tablas que se rellenaban en el script *transcription.sh*:

- *conversations* almacena el nombre de la conversación recibido como argumento e incluye un campo para añadir una descripción textual y la fecha de transcripción.
- *streams* almacena información sobre los ficheros asociados a la conversación, asumiendo que se encuentran en un directorio predeterminado. Se trata de cuatro entradas asociadas a la entrada de la tabla *conversations*, cada una con los campos *nombre*, *tipo* y *alias*:
  - nombre *nombre_hablante1*, tipo audio y alias por defecto *nombre_hablante1*.
  - nombre *nombre_hablante2*, tipo audio y alias por defecto *nombre_hablante2*.
  - nombre *nombre_ conversación*, tipo audio y alias vacío.
  - nombre *nombre_ conversación*, tipo texto y alias vacío.

El campo alias está pensado para permitir al analista usar nombres que sean más cómodos para él en lugar de las direcciones manejadas por los protocolos (una dirección SIP de ejemplo sería user1@here.com, direcciónIP@here.com).

El primer problema que se planteó fue relacionar las entradas de la tabla *conversations* con la correspondiente entrada del contenido SIP (*Session Initiation Protocol*) o RTP (*Real-time Transport Protocol*). La decisión tomada fue añadir en las tablas *sips* y *rtps* un campo que apuntara a la conversación.

Posteriormente se realizaron pequeñas modificaciones manteniendo la estructura general de las dos tablas, simplemente añadiendo algunos campos útiles. Sin embargo, tiempo después se decidió afrontar una remodelación total, almacenando toda la información en la tabla *conversations* y eliminando la tabla *streams*. La tabla *conversations* ha quedado como se indica a continuación:

- *Transcription_filepath*: ruta al fichero con la transcripción textual.
- *Transcription_date*: fecha en que se realizó la transcripción.
- *Voip_table*: nombre de la tabla que contiene el audio (de momento: *sips*,

rtps).

- *Voip_id*: identificador del audio dentro de la tabla *voip_table*.
- *Description.*
- *Conversation_name*: nombre de la conversación., generado por Xplico.
- *Conversation_alias*. Permite definir un alias para la conversación.
- *Conversation_filename*: nombre del fichero con el audio de la conversación.
- *Conversation_filepath*: ruta al fichero con el audio de la conversación.
- *Speaker_from_address*: dirección de correo del hablante que inició la conversación.
- *Speaker_from_alias*: alias del hablante que inició la conversación.
- *Speaker_from_filename*: nombre del fichero con el audio del hablante que inició la conversación.
- *Speaker_from_filepath*: ruta al fichero anterior.
- *Speaker_to_address* dirección de correo del hablante que recibió la conversación.
- *Speaker_to_alias:* alias del hablante que recibió la conversación.
- *Speaker_to_filename* nombre del fichero con el audio del hablante que recibió la conversación.
- *Speaker_to_filepath*: ruta al fichero anterior.

En cuanto a los scripts, lo primero que hubo que cambiar es que estos sólo recibían como argumento el nombre de los ficheros audio asumiendo que su nombre coincidía con el del hablante y que estaban almacenados en una carpeta predeterminada. Esto obviamente supone una limitación y no concuerda con el sistema de nombres de ficheros de *Xplico* que genera los nombres según el tipo de contenido y la fecha y hora en que se obtuvo. Además los scripts *files.sh* y *training.sh* no necesitaban el fichero de audio sino los nombres de los hablantes a los que generar el sistema de ficheros y el nombre del hablante para el que realizar el entrenamiento, respectivamente. Es decir, en todos los scripts hubo que modificar los argumentos y variables internas necesarias y las llamadas a comandos que hacían uso de ellas. Por otro lado, también se asumía que siempre hay presentes dos ficheros de audio correspondientes a dos interlocutores pero en la práctica puede suceder que sólo haya audio de uno de ellos, por lo que es necesario considerar esta opción también.

La nueva sintaxis de los argumentos de los scripts se reduce a:

- *audioSplit.sh <conversation.txt> <pad> <user1_name> <user1.wav> [<user2.wav> <user2_name>]*
- *files.sh <conversation.txt> <user1_name> [<user2_name>]*
- *timestamps.sh <user.wav> <user_name> <fragment.wav> <conversation.txt> <line_index> <old_timestamp> <new_timestamp>*
- *training.sh <user_name>*

- *transcript.sh <conversation_name> <user_from_name> [-f <user_from_file>] <user_to_name> [-f <user_to_file>]*

El formato de los ficheros de audio también provocó otros cambios en el script *transcript.sh*. *Xplico* decodifica el audio mediante la herramienta videosnarf, versión 0.63 para ser más concretos. Esta herramienta genera ficheros *.wav* pero *Xplico* los convierte a *.mp3* para ahorrar espacio. Es posible configurar *Xplico* para que borre o no los ficheros *.wav* originales por lo que se pueden pasar directamente los ficheros *.wav* y evitar realizar de nuevo una conversión a *mp3*, lo que de hecho mejora notablemente la calidad del audio que llega al módulo de transcripción. Sin embargo, mediante pruebas se observó que usar los *.wav* originales daba errores y era necesario realizar la conversión de frecuencia en cualquier caso. La conclusión es que ya que requieren el mismo procesamiento, el script puede aceptar tanto *.wav* como *.mp3* pero debido a la clara diferencia en la calidad del audio, decidimos utilizar los *.wav* originales. Para no hacer uso de excesiva memoria innecesariamente y ya que es necesario mantener una copia del *.wav* generado para poder entrenar el sistema, el script genera un nuevo *.wav* con la nueva frecuencia en la carpeta predeterminada del módulo de transcripción y borra el fichero *.wav* original, manteniendo la copia *mp3* generada por *Xplico*.

Otro de los detalles importantes a mejorar era que en este mismo script, *transcript.sh*, cuando existía modelo acústico adaptado al usuario, lo que hacía era, renombrar el modelo genérico *hub.jar* a *hub4.temp*, renombrar el modelo adaptado a hub.jar, utilizarlo para la transcripción y renombrar ambos ficheros a sus nombres originales. Investigando en el porqué, se descubrió que el problema radicaba en la forma de empaquetar el fichero *.jar*, puesto que en el fichero de configuración XML se indicaba que el modelo acústico estaba bajo *hub4/*, pero no existía dicho directorio en el fichero *jar* del modelo adaptado. Simplemente modificando la estructura generada en el *script* de entrenamiento *training.sh* para que se sitúe bajo un directorio con este nombre, permitió evitar el innecesario renombramiento de los ficheros. Además se han movido varios directorios desde el directorio *lib/* de Sphinx al directorio */opt/* para mantener separados y fácilmente accesibles los modelos acústicos generados por el modulo.

En cuanto al fichero *timestamps.sh* ahora recibe como parámetro el índice del fragmento que hay que modificar lo que permite que las marcas de tiempo no tengan que ser únicas.

La interfaz web, que también estaba desarrollada en *CakePHP* para facilitar la integración con *Xplico*, se adaptó al estilo de la aplicación web de *Xplico*, con algunos cambios respecto a la organización original. En este punto se observaron algunos detalles que fue necesario tratar. Al igual que en el caso del *Plugin Manager*, este módulo se desarrolló con idea de que la integración se realizara posteriormente aunque no estaba totalmente preparado para ella. La funcionalidad de entrenamiento de la interfaz hacía uso de contenidos almacenados en una carpeta dentro del sistema de ficheros de la propia aplicación web (directorio *webroot/* para ser exactos) en lugar de dentro de la estructura de ficheros del módulo de transcripción propiamente dicho. Una vez que se empezaron a hacer los cambios pertinentes, empezó a haber problemas de permisos o de que la interfaz no era capaz de acceder a los ficheros de audio por problemas de rutas. El primer caso se daba, por ejemplo, durante el entrenamiento. El script *training.sh* crea varios ficheros en el directorio que se le indique, que resultó ser el directorio actual, por lo que dependía del lugar desde que se llamara al *script*. Al ser invocado desde la interfaz web, el directorio local pertenecía a la estructura de

directorios de la aplicación web, lo que daba problemas de acceso. Los problemas de rutas surgían porque la estructura de ficheros del módulo de transcripción está fuera del de la aplicación web, por lo que eran inaccesibles por esta. La solución que se adoptó fue la más sencilla: ya que se trataba de dos carpetas concretas, se crearon enlaces simbólicos a las mismas dentro de la estructura de ficheros de la aplicación web.

Aparte, se modificaron tanto controladores como vistas, cambiando totalmente el procesamiento interno, haciendo un código mucho más sencillo. El resultado final se muestra en las Figuras 35 y 36, capturas de la interfaz del módulo VOIP ya integrada totalmente, accesible desde las vistas de SIP y RTP como se aprecia en la Figura 37. De momento se permite también visualizar la lista de transcripciones, mediante el enlace *Transcriptions* que se aprecia en el menú desplegable de la izquierda.

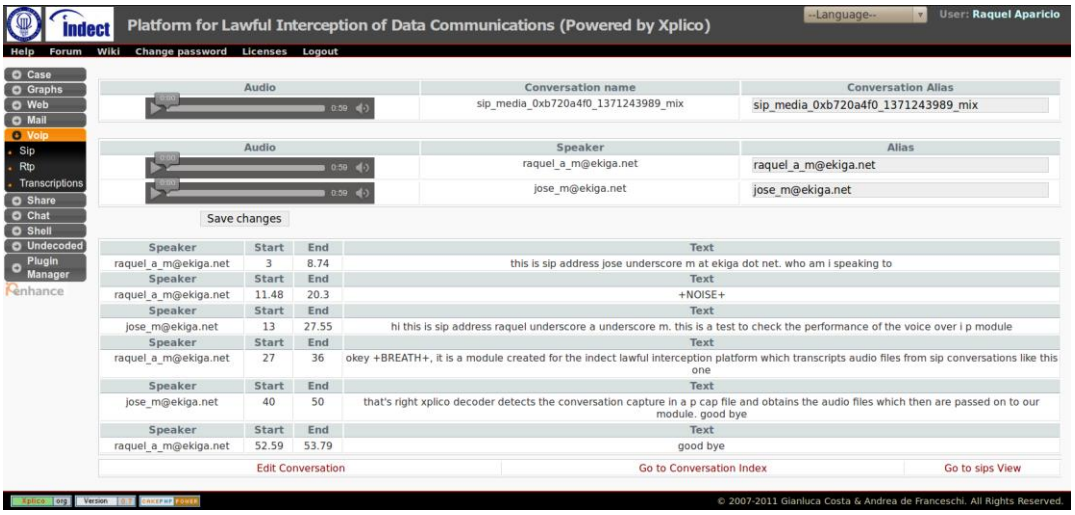


Figura 35. Captura de pantalla de la visualización de una transcripción.

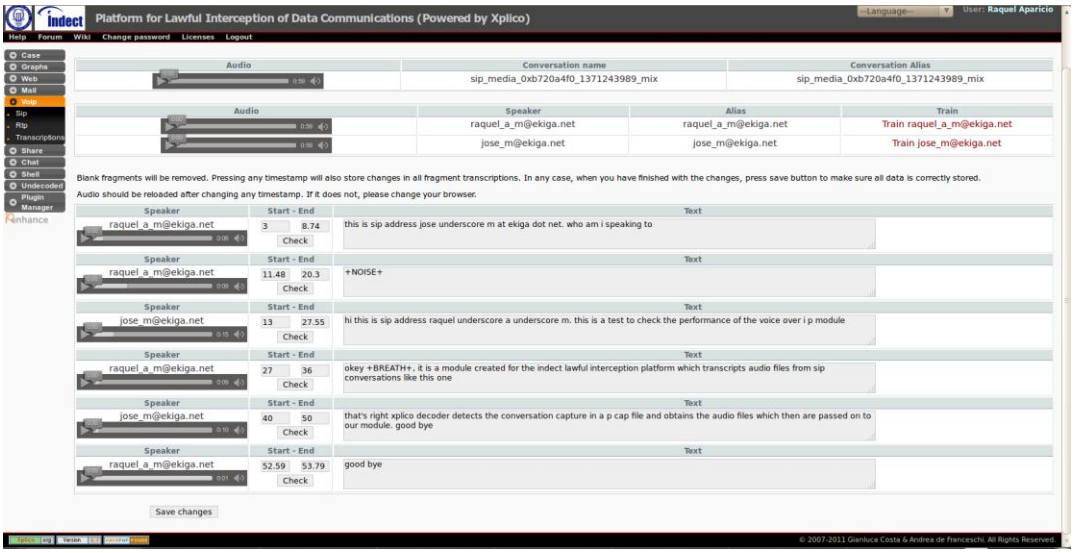


Figura 36. Captura de pantalla de la edición de una transcripción.

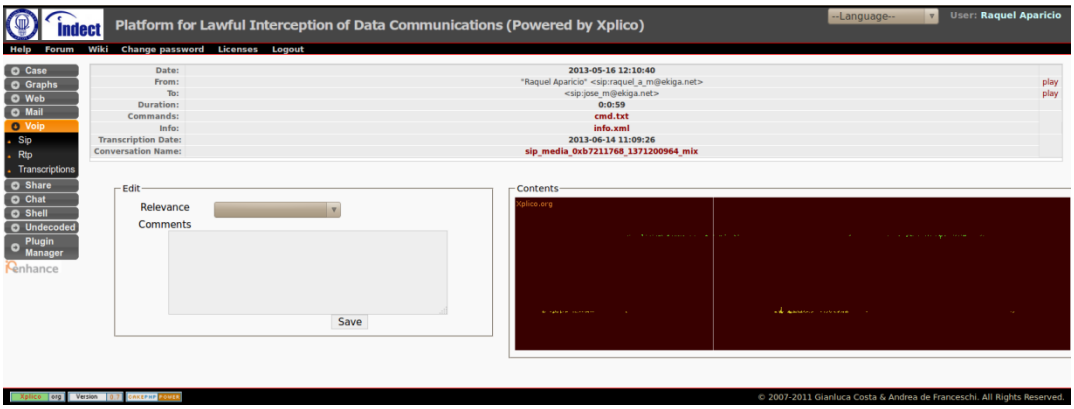


Figura 37. Captura de pantalla de la visualización de una conversación.

Capitulo 4: **Evaluación.**

En este apartado veremos distintas pruebas que se han realizado sobre la plataforma a lo largo del proyecto, en el mismo orden en que se ha explicado la integración. Primero se presentará la información de forma visual en una tabla y a continuación se detallará un poco más la información sobre las pruebas.

Prueba	**Resultado**	**¿Es correcto? ¿Solución?**
Procesamiento de contenidos cifrados.	*Xplico* no es capaz de decodificarlos.	No es un error. No es posible decodificar contenidos cifrados.
Intento de conexión mediante certificado digital: 1. Completo (con UID), usuario autorizado. 2. Completo (con UID), usuario no autorizado. 3. Incompleto (sin UID), usuario autorizado. 4. Incompleto (sin UID), usuario no autorizado. 5. Caducado 6. Creado por otra PKI	1. Acceso permitido automáticamente. Resto: error y opción de identificarse mediante nombre y contraseña.	Correcto.
Uso de dispositivos de seguridad (*tokens*).	Firefox se bloquea bajo Linux.	Fallo de Firefox. Utilizar otro navegador o Windows.
Servidor LDAP: 1. Usuario no presente en el sistema. 2. Usuario presente pero no autorizado. 3. Usuario presente y autorizado.	1. No existe el usuario. 2. El usuario no tiene autorización. 3. El usuario tiene autorización	Correcto.
Carga de un fichero de contenidos de gran tamaño	Sin problemas de memoria.	Correcto.
Ejecución de *plugins* locales.	Comunicación correcta.	Correcto.
Ejecución de *plugins* remotos.	Comunicación correcta.	Correcto.
Transcripción de contenidos VoIP.	Comunicación correcta pero resultado con baja calidad.	No es un error. Con entrenamiento se mejora la transcripción.

Al hablar de *Xplico* se mencionó su limitación en cuanto a contenidos cifrados: no es capaz de procesarlos. Esto se puede observar sencillamente al conectarse a sitios web mediante HTTPS en lugar de HTTP, realizando una captura en vivo durante la que se realicen movimientos mediante ambos protocolos. En el caso de servidores de correo

online, como son Gmail, Hotmail o Yahoo, Gmail y Hotmail obligan a utilizar HTTPS pero Yahoo no. Haciendo unas sencillas pruebas de envío de correo entre distintas cuentas, se observa que *Xplico* sólo decodifica lo enviado y recibido a través de Yahoo. Por ejemplo, al hacer una captura de tráfico mientras se mantiene una conexión online a Gmail y Yahoo, si se envían correos desde ambas cuentas y se visualiza en ambas tanto la bandeja de entrada como correos concretos, nada de lo hecho a través del sitio web de Gmail aparece decodificado en *Xplico*, pero sí lo hecho a través de Yahoo:

- Un email enviado de Gmail a Gmail no aparece entre los contenidos.
- Un email enviado de Gmail a Yahoo aparece al visualizarlo en Yahoo.
- Un email enviado de Yahoo a Gmail aparece como enviado desde Yahoo pero no tras visualizarlo en Gmail.
- Un email enviado de Yahoo a Yahoo aparece tanto como enviado como al visualizarlo en Yahoo.

En cuanto a los certificados digitales, se comprobó que el funcionamiento de la plataforma era el correcto, para lo cual se realizaron pruebas con certificados:

- Que contenían el campo identificador de usuario (UID), tanto para usuarios autorizados como no autorizados.
- Que no lo contenían, tanto para usuarios autorizados como no autorizados.
- Que estaban caducados.
- Creados por una jerarquía de CAs distinta.

Los certificados caducados o los creados por otra jerarquía son rechazados por el servidor Apache directamente si se utiliza la opción *SSL_Client_Verify optional*. En este caso, no se llega a conectar con la aplicación web en ningún caso. Por el contrario, si se utiliza la opción *SSL_Client_Verify optional_no_CA*, sí se produce una conexión a la aplicación web, que comprueba la validez de los certificados y pide autenticación mediante usuario y contraseña si no lo son.

Durante las pruebas con los dispositivos de seguridad, es decir, los lectores de tarjetas inteligentes, se observó que bajo el sistema operativo Ubuntu el navegador Firefox se queda bloqueado tras el envío del certificado al servidor y sólo se desbloquea al desconectar el dispositivo, realizando entonces la autorización y mostrando la pantalla principal de la herramienta. No fue posible utilizar el lector con otros navegadores como Chrome u Opera puesto que no ofrecen la posibilidad de configurar el dispositivo. Sin embargo, se realizaron pruebas con Firefox desde Windows y no se observó el problema.

En lo que respecta al servidor LDAP, la primera intención fue crear un servidor seguro, que recibiera conexiones mediante LDAPS o mediante el uso del comando *startTLS*. Sin embargo, tras muchos intentos y dado que OpenLDAP está compilado contra la herramienta GnuTLS, no fue posible lograrlo debido a las diferencias entre esta herramienta y OpenSSL. Se procedió a intentarlo con certificados generados por la herramienta *certtool* de GnuTLS, pero incluso así la conexión seguía siendo rechazada, por lo que se decidió mantener el uso de la herramienta OpenSSL puesto que el servidor LDAP estará situado en el mismo *data center* que el resto, y su nivel de protección será

considerable. Las pruebas que se realizaron sobre el servidor LDAP fueron similares a las de HTTPS: usuarios presentes en el sistema, tanto autorizados (usuario normal o administrador) como no autorizados y usuarios que no existían en el sistema. Los resultados fueron los esperados: sólo se permite entrar a los usuarios presentes en el sistema que además están autorizados para entrar en la aplicación.

En cuanto a los problemas que aparecieron respecto a los hilos que lanzaban las distintas partes del sistema, se hicieron varias pruebas para comprobar qué tamaño seria el adecuado para la piscina de hilos del módulo distribuidor de contenidos, de forma que no se sobrecargara o incluso bloqueara el sistema. Se decidió darle un valor de un máximo de 100 hilos ejecutando *plugins* simultáneamente. Posteriormente se eliminó dicha piscina de hilos ofreciendo una funcionalidad síncrona para aprovechar la piscina de hilos del propio *Xplico*. Debido a que el acceso a la base de datos por parte del módulo distribuidor no estaba sincronizado con el de *Xplico*, reiteradamente aparecían mensajes de error acerca de que estaba ocupada. Inicialmente se creó un *mutex* interno para el módulo pero después se decidió permitir que *Xplico* enviara su propio *mutex* de manera que la sincronización sea total. El rendimiento mejoró notablemente.

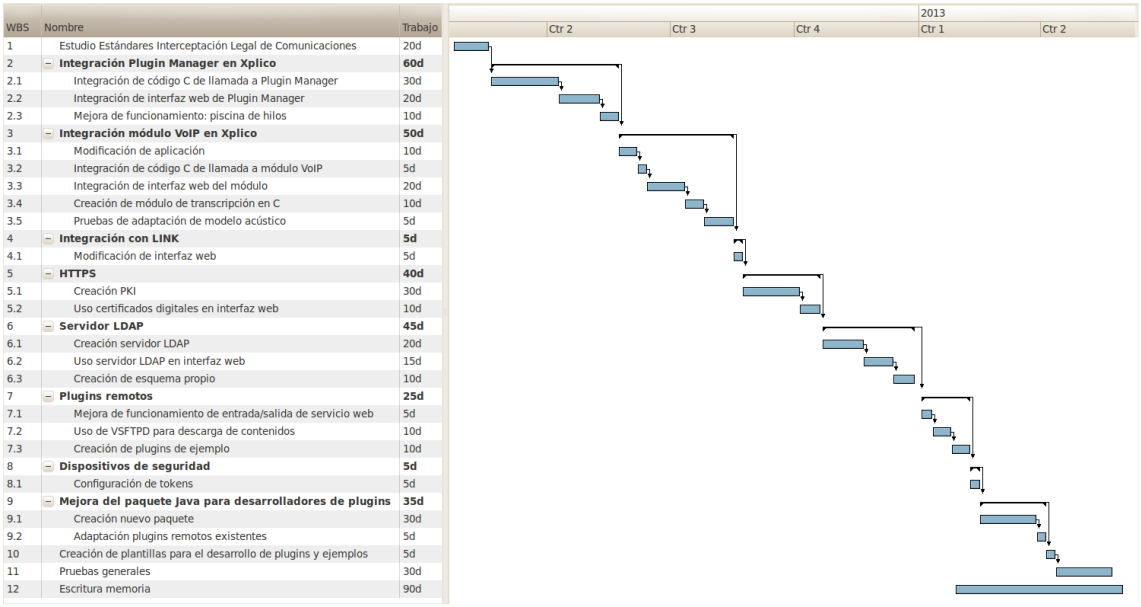
Los *plugins* remotos se probaron mediante varios *plugins* sencillos. Se partió de un *plugin HelloWorld* al que se fue añadiendo más funcionalidad para comprobar su rendimiento. Gracias a esto se descubrieron y solucionaron varios fallos que tenía la interfaz de web services. Posteriormente, se creó un *plugin* remoto basado en la herramienta *SpamAssassin* que permite saber si un contenido es *spam* o no. La propia herramienta tiene un modo de entrenamiento mediante el comando *sa-learn*. Se observó que para entrenar correctamente esta herramienta es necesario disponer de mucho material tanto de *spam* como de no *spam*.

A lo largo de las pruebas con los *plugins* remotos se realizaron también las pruebas con el servidor FTP, para mejorar la comunicación sin necesidad de que el módulo distribuidor envié el usuario y contraseña en claro con la URL del contenido. Se hicieron pruebas para aumentar la seguridad lo máximo posible, permitiendo el acceso sólo a usuarios virtuales definidos en un determinado fichero, y restringiendo sus permisos de forma que no pueden ni listar ni escribir, ni tampoco salirse de la carpeta */opt/*. También se limitó el tiempo de conexión inactivo. Todas estas decisiones se basan en que un *plugin* remoto conectara únicamente para bajarse un contenido concreto, por lo que no necesita ni otros permisos ni tiempo de inactividad. Además, en el uso de la librería *Apache Commons Net*, permite un mayor control sobre la interacción y abre las puertas a realizar una conexión FTPS en el futuro.

Por otro lado, las pruebas que se realizaron con el módulo VoIP demostraron que necesita entrenamiento ya que las transcripciones textuales resultaron ser bastante malas. Entrenando un modelo con unas pocas grabaciones de mi voz mejoraba las transcripciones de las conversaciones audio capturadas pero, puesto que otras tareas eran más importantes, se decidió dejar como tarea futura la preparación de una gran cantidad de datos de audio transcritos para realizar un buen entrenamiento

Capitulo 5: **Presupuesto**

En este apartado veremos cuál es el presupuesto del proyecto tanto en términos de recursos humanos como en recursos materiales. En su desarrollo han participado dos ingenieros: uno *Junior* como desarrollador y uno *Senior* como coordinador. La duración del proyecto ha sido aproximadamente de 18 meses, desde Enero de 2012 hasta Junio de 2013. El desglose del proyecto en sus distintas fases se presenta en la Figura 38.



WBS	Nombre	Trabajo
1	Estudio Estándares Interceptación Legal de Comunicaciones	20d
2	**Integración Plugin Manager en Xplico**	**60d**
2.1	Integración de código C de llamada a Plugin Manager	30d
2.2	Integración de interfaz web de Plugin Manager	20d
2.3	Mejora de funcionamiento: piscina de hilos	10d
3	**Integración módulo VoIP en Xplico**	**50d**
3.1	Modificación de aplicación	10d
3.2	Integración de código C de llamada a módulo VoIP	5d
3.3	Integración de interfaz web del módulo	20d
3.4	Creación de módulo de transcripción en C	10d
3.5	Pruebas de adaptación de modelo acústico	5d
4	**Integración con LINK**	**5d**
4.1	Modificación de interfaz web	5d
5	**HTTPS**	**40d**
5.1	Creación PKI	30d
5.2	Uso certificados digitales en interfaz web	10d
6	**Servidor LDAP**	**45d**
6.1	Creación servidor LDAP	20d
6.2	Uso servidor LDAP en interfaz web	15d
6.3	Creación de esquema propio	10d
7	**Plugins remotos**	**25d**
7.1	Mejora de funcionamiento de entrada/salida de servicio web	5d
7.2	Uso de VSFTPD para descarga de contenidos	10d
7.3	Creación de plugins de ejemplo	10d
8	**Dispositivos de seguridad**	**5d**
8.1	Configuración de tokens	5d
9	**Mejora del paquete Java para desarrolladores de plugins**	**35d**
9.1	Creación nuevo paquete	30d
9.2	Adaptación plugins remotos existentes	5d
10	Creación de plantillas para el desarrollo de plugins y ejemplos	5d
11	Pruebas generales	30d
12	Escritura memoria	90d

Figura 38. Diagrama de Gantt del proyecto.

En cuanto al coste económico, éste se divide por un lado en los sueldos de los ingenieros y por otro en el coste del equipamiento. Para definir los sueldos de los ingenieros partiremos de un salario base de 2.500 euros mensuales para un ingeniero Junio. En el caso del ingeniero *Senior* este salario base se ve incrementado a digamos 4.000 euros mensuales. La dedicación del ingeniero Junior ha sido total durante la duración del proyecto, es decir, 18 hombres/mes, mientras que la del Ingeniero *Senior* se ha debido a la organización por lo que se puede reducir a 2 hombre/mes. El presupuesto dedicado a salarios queda entonces como:

Tipo empleado	**Hombres/mes**	**Sueldo al mes**	**Sueldo total**
Ingeniero Junior	18	2.500 euros	45.000 euros
Ingeniero Senior	2	4.000 euros	8.000 euros
		Total	53.000 euros

El equipamiento necesario durante el proyecto ha sido:

Tipo	Desglose	Precio	Uso en el proyecto	Dedicación	Periodo depreciación	Coste al proyecto
Hardware	Ordenador de sobremesa	1.000 euros	100%	18 meses	60 meses	300 euros
	Dispositivos de seguridad	90 euros		2 meses		3 euros
Software	Ubuntu	0 euros	100%	18 meses		0 euros
	VirtualBox					
	CakePHP					
	Xplico					
	SQLite					
	Apache					
	OpenLDAP			7 meses		
	VSFTPD			5 meses		
					Total	303 euros

Si sumamos los costes tanto humanos como materiales, obtenemos la suma de 53.303 euros. Si además consideramos que la tasa de costes indirectos es de un 20%, el resultado final asciende a 63.964 euros.

Tras todos lo detallado hasta el momento, podemos resumir que el presupuesto total de este proyecto asciende a la cantidad de 63.964 euros.

Leganés a 14 de Junio de 2013

El ingeniero proyectista

Fdo. Raquel Aparicio Morenilla

Capitulo 6: **Conclusiones y trabajos futuros**

El objetivo de este proyecto fin de carrera es la integración de distintas herramientas, en forma de módulos y aplicaciones, para la creación de una única plataforma encargada de procesar las comunicaciones interceptadas a un sujeto bajo investigación. Recordemos el funcionamiento final de dicha plataforma para poder comprobar qué puntos se han podido cumplir y que tareas quedan pendientes. Las comunicaciones del sujeto bajo investigación se proporcionan a la plataforma en forma de ficheros de captura *.pcap*, que son analizados por la aplicación *Xplico* para obtener las comunicaciones propiamente dichas, en forma de ficheros de contenidos. Posteriormente, la integración realizada permite que estos contenidos sean pasados de forma automática a distintos módulos o herramientas, según el tipo de contenido. Un contenido de tipo audio, es pasado primero al módulo de transcripción VoIP y una vez se ha obtenido su transcripción es pasado al módulo *Plugin Manager*. Otro tipo de contenidos son pasados directamente al *Plugin Manager*. Este módulo se encarga de distribuir los contenidos a herramientas capaces de procesar su tipo, denominadas *plugins*. El procesamiento de distintos contenidos se realiza en paralelo, habiendo sido necesario modificarlo para utilizar una piscina de hilos que permita limitar el consumo de memoria global. Anteriormente a realizar la llamada a los *plugins*, el *Plugin Manager* se encarga de completar la información recibida de la herramienta *Xplico* para aportarles los datos necesarios. El procesamiento realizado por los *plugins* da como resultado una preclasificación del contenido, en forma de un número del 1 al 5 y una descripción textual acerca del resultado o el procesamiento. Si el *plugin* no es capaz de dar una preclasificación devolverá un 0 y si ha surgido un error durante el procesamiento, un -1. Cada vez que un *plugin* termina su funcionamiento, el *Plugin Manager* se encarga de comprobar si es necesario continuar con el procesamiento del contenido, en caso de que se haya producido un error o no haya sido posible realizar la pre-clasificación, o por el contrario se puede devolver el resultado a *Xplico*, cuyo código fue modificado para recibir este resultado y actualizar la información en la base de datos.

Además, la aplicación presenta una interfaz web para un fácil acceso a los contenidos y a la configuración del *Plugin Manager* y los *plugins*. Esta interfaz es fruto de la fusión de las interfaces de *Xplico*, de la del *Plugin Manager*y los *plugins* y la del módulo de transcripción VoIP. También fue necesario fusionar las bases de datos de estas tres herramientas, para así centralizar la información.

El problema básico a lo largo de todo el proyecto fue el manejo de código desarrollado por diferentes personas y en distintos lenguajes de programación: Java, C, C++ PHP, Javascript, scripts. Las herramientas utilizadas estaban destinadas a ser integradas pero los desarrolladores no tenían conocimiento de los detalles necesarios acerca de las demás, por lo que a la hora de la verdad fueron desarrolladas de forma totalmente independiente. Ésto implicó que la mayoría de los problemas y modificaciones realizadas tenía que ver con decisiones inadecuadas a la realidad, algunas de las cuales no era posible prever en su momento. Sin embargo, precisamente esta gran variedad de lenguajes de programación y tecnologías empleadas (certificados digitales, LDAP, servidor Apache, web services) hace el proyecto mucho más interesante.

En cuanto a tareas pendientes, existen trabajos futuros en varios flancos:

- La integración con el resto de herramientas disponibles y con las que se vayan desarrollando. En particular, queda pendiente la integración del módulo de identificación de contenidos basado en *hashes* difusos que ya está disponible.
- El desarrollo de *plugins* que amplíen las posibilidades de la plataforma.
- La mejora del módulo de transcripción mediante su entrenamiento para usuarios puesto que se necesita una gran cantidad de material auditivo de cada usuario.
- Incremento de la seguridad, incluso de forma interna al sistema INDECT, mediante el uso del protocolo LDAPS y comunicaciones seguras bajo Axis2 entre el *Plugin Manager* y los *plugins* remotos en forma de web services, para lo cual es necesario de nuevo el uso de certificados digitales.

Referencias

[1] Página web oficial del proyecto INDECT. Disponible [Internet]: <http://www.indect-project.eu/> [27 de Noviembre de 2012].

[2] Página oficial de Redtel. Disponible [Internet]: <http://redtel.es> [27 de Noviembre de 2012].

[3] Alegre Ramírez-Montesinos, Isaac. *Interceptación Legal de Comunicaciones IP.* Mundo Internet. XII Congreso Iberoamericano de Internet, Telecomunicaciones y Sociedad de la información. 2009. Disponible [Internet]: <http://www.mundointernet.es/IMG/pdf/ponencia57.pdf> [27 de Noviembre de 2012].

[4] Página web oficial de la herramienta *Xplico*. Disponible [Internet]: <http://xplico.org> [14 de Junio de 2013].

[5] Página oficial del proyecto LINK de la universidad de AGH. Disponible [Internet]: <https://link.iisg.agh.edu.pl/> [27 de Noviembre de 2012].

[6] Rubén Mena Escribano. "Módulo de distribución y pre-clasificación de contenidos". Proyecto fin de carrera de la Universidad Carlos III de Madrid.

[7] Jesús Vallinot Sánchez. "Módulo de transcripción VoIP-texto". Proyecto fin de carrera de la Universidad Carlos III de Madrid.

[8] Roberto Sánchez. "Módulo de clasificación de contenidos basado en hashes difusos". Proyecto fin de carrera de la Universidad Carlos III de Madrid.

[9] Herramienta de procesado de imágenes INACT (*INDECT Advance Image Catalog Tool*), desarrollada en el grupo de trabajo WP5.

[10] ETRI ETR 331: *Security Techniques Advisory Group (STAG); Definition of user requirementes for lawful interception of telecommunications; Requirements of the law enforcement agencies*. Diciembre de 1996.

[11] European Union Council Resolution on the Lawful Interception of Telecommunications. 17 Enero 1995.

[12] ETRI ETR 330: *Security Techniques Advisory Group (STAG); A guide to legislation, recommendations and guidelines governing the provision of security features*. Primera edición, noviembre 1996.

[13] ETRI ES 201 158: *Telecommunications security; Lawful Interception (LI); Requirements for network functions.* V1.1.2, mayo de 1998.

[14] ETRI ES 201 671: *Telecommunications security; Lawful Interception (LI); Handover interface for the lawful interception of telecommunications traffic*. V1.1.1, julio de 1999.

[15] ETRI TS 101 331: *Lawful Interception (LI); Requirementes of Law Enforcement Agencies.* Versión 1.3.1. Octubre de 2009.

[16] ETRI ES 201 158: *Telecommunications security; Lawful Interception (LI); Requirements for network functions.* Versión 1.2.1. Abril de 2002.

[17] ETRI TS 101 671: *Telecommunications security; Lawful Interception (LI); Handover interface for the lawful interception of telecommunications traffic*. Versión 3.10.1. Junio de 2012.

[18] ETRI TR 101 943: *Lawful Interception (LI); Concepts of Interception in a Generic Network Architecture.* Versión 2.2.1. Noviembre de 2006.

[19] ETRI TR 101 944: *Telecommunications security; Lawful Interception (LI); Issues on IP interception.* Versión 1.1.2. Diciembre de 2001.

[20] ETRI TR 102 528: *Lawful Interception (LI); Interception domain Architecture for IP networks*. Versión 1.1.1. Octubre de 2006.

[21] ETRI TS 102 232-1: *Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 1: Handover specification for IP delivery.* Versión 3.1.1. Junio de 2012.

[22] ETRI TS 102 232-2: *Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 2: Service-specific details for messaging services*. Versión 3.1.1. Febrero de 2012.

[23] ETRI TS 102 232-3: *Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 3: Service-specific details for internet access Services.* Versión 3.2.1. Junio de 2012.

[24] ETRI TS 102 232-4: *Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 4: Service-specific details for Layer 2 Services.* Versión 3.1.1. Febrero de 2012.

[25] ETRI TS 102 232-5*: Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 4: Service-specific details for IP Multimedia Services*. Versión 3.2.1. Junio de 2012.

[26] ETRI TS 102 232-6: *Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part7: Service-specific details for PSTN/ISDN services.* Versión 3.1.1. Junio de 2012.

[27] ETRI TS 102 232-7: *Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part7: Service-specific details for Mobile Services*. Versión 3.1.1. Junio de 2012.

[28] Página oficial de CakePHP. Disponible [Internet]: <http://cakephp.org> [27 de Noviembre de 2012].

[29] Golding, David. *Beginning CakePHP: From Novice to Profesional.* Apress. ISBN: 978-1-4302-09775. 2008.

[30] Poster del proyecto LINK de la universidad AGH. Disponible [Internet]: <home.agh.edu.pl/~dajda/link-poster.pdf> [27 de Noviembre de 2012].

[31] Open LDAP OID Registry. Disponible [Internet]: <www.openldap.org/faq/data/cache/197.html> [14 de Junio de 2013].

[32] Tipos de datos en LDAP. Disponible [Internet]: <http://zytrax.com/books/ldap/apa/types.html> [14 de Junio de 2013].

[33] Página web de CMU Sphinx. Disponible [Internet]: <http://cmusphinx.sourceforge.net > [14 de Junio de 2013].

[34] Reconocimiento automático del habla. Disponible [Internet]:

<http://dihana.cps.unizar.es/investigacion/voz/rahframe.html> [27 de Noviembre de 2012].

[35] Guía de uso de dispositivos inteligentes de Gooze. Disponible [Internet]: <http://www.gooze.eu/howto/smartcard-quickstarter-guide> [14 de Junio de 2013].

[36] Página oficial de Ubuntu. Disponible [Internet]: <https://help.ubuntu.com/12.04/serverguide/openldap-server.html> [27 de Noviembre de 2012].

[37] Página web de zyntrax. Disponible [Internet]: < www.zytrax.com/books/ldap > [27 de Noviembre de 2012].

[38] Página web de linuxtopia. Disponible [Internet]: <http://www.linuxtopia.org/online_books//network_administration_guides/ldap_administration/slapdconf2_Configuration_Directives.html> [27 de Noviembre de 2012].

[39] Página web acerca de certificados digitales. Disponible [Internet]: <http://www.uv.es/~sto/articulos/BEI-2003-11/certificados_digitales.html>

[40] X.509: Information technology - Open systems interconnection - The Directory: Public-key and attribute certificate frameworks. Octubre de 2012.

[41] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Mayo de 2008.

[42] *RFC 3739: Internet X.509 Public Key Infrastructure: Qualified Certificates Profile*. Marzo de 2004.

[43] Página web de openssl.org. Disponible [Internet]: <http://www.openssl.org/docs/apps/ca.html > [14 de Junio de 2013].

[44] Página web de openssl.org. Disponible [Internet]: <http://www.openssl.org/docs/apps/req.html> [14 de Junio de 2013].

[45] Página web de openssl.org. Disponible [Internet]: <http://www.openssl.org/docs/apps/x509v3_config.html> [14 de Junio de 2013].

Anexo I: CakePHP

CakePHP [28] es un marco de trabajo para aplicaciones web que utiliza el lenguaje PHP y el patrón Modelo-Vista-Controlador (MVC) donde:

- El modelo es la representación específica de la información que el sistema maneja, los datos de la aplicación.
- La vista es la presentación de los datos, la interfaz de usuario.
- El controlador es la lógica de la aplicación y responde a eventos (acciones del usuario) e invoca peticiones al modelo y la vista.

Este patrón permite crear aplicaciones modulares fáciles de mantener puesto que las tres tareas que acabamos de mencionar permanecen separadas físicamente en distintos ficheros y claramente organizadas en directorios. La aplicación es fácil de entender y modificar, tanto para añadir nuevas características como para modificar las existentes sin que el resto se vea afectado.

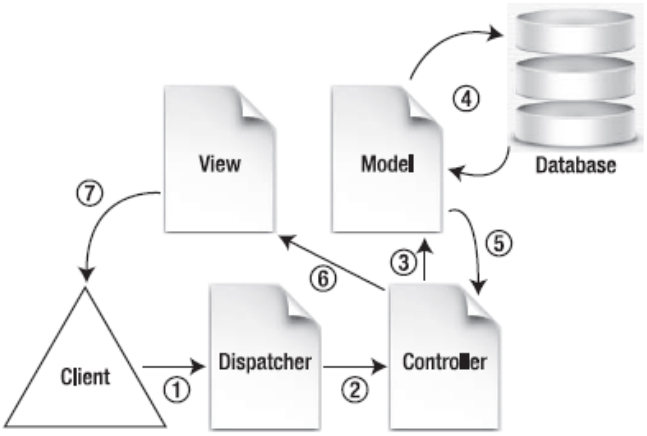


Figura 39. Patrón Modelo-Vista-Controlador en CakePHP [29].

Los pasos del proceso MVC dependen del marco de trabajo pero en general son los mostrados en la Figura 39:

1. El cliente manda una petición a la aplicación bien pulsando un enlace o bien tecleando una URL que normalmente sigue el formato:

   http://{Domain}.com/{Application}/{Controller}/{Action}/{Parameter 1, etc.}

2. El script *dispatcher* de la figura procesa la URL para determinar qué controlador ejecutar. Después invoca dicho controlador indicando qué acciones ejecutar y con qué parámetros.
3. Si la función del controlador necesita más datos que los pasados por parámetro realiza peticiones al script del modelo.
4. El script del modelo determina cómo interactuar con la base de datos para realizar las peticiones recibidas.
5. El modelo devuelve los datos al controlador.
6. El controlador procesa los datos y envía lo necesario a la vista.
7. La vista añade información de diseño y formato a los datos recibidos y lo envía al cliente.

CakePHP proporciona la funcionalidad de los scripts mencionados en estos pasos, por lo que sólo es necesario que sea capaz de relacionar las distintas piezas entre sí. Para eliminar la necesidad de que el programador tenga que indicarlo manualmente, el marco de desarrollo propone una nomenclatura que le permite encontrar las relaciones automáticamente. Es lo que se denomina convención sobre configuración y se explicará a continuación, al comentar la estructura de ficheros de una aplicación basada en CakePHP.

Una vez que se tiene un servidor web funcionando y la base de datos de la aplicación, el siguiente paso es descargar y descomprimir la última versión de CakePHP de *http://github.com/cakephp/cakephp/downloads* y situar el directorio obtenido bajo la ruta del servidor web (*DocumentRoot*). Bajo dicho directorio se sitúan varios ficheros y subdirectorios de los que nos interesa *app*. Éste contiene entre otras:

- Un directorio *controllers/* que almacena los ficheros *.php* que contienen los controladores.
- Un directorio *models/* que almacena los ficheros *.php* que contienen los modelos.
- Un directorio *views/* que almacena los ficheros *.ctp* que contienen las vistas.
- Un directorio *webroot/* que almacena imágenes u otros ficheros para incluir en las vistas, scripts en código javascript, ficheros *.css* que definen estilos para las vistas…
- Un directorio *config/* que almacena la configuración de la aplicación web, como por ejemplo el fichero database.php, que contiene la información de la base de datos a utilizar: tipo, usuario y contraseña, localización, etc.
- Un directorio *plugins/* que almacena otras aplicaciones de las que la aplicación que estamos creando vaya a hacer uso.

Son las tres primeras carpetas las que contienen la estructura de la aplicación: modelos, vistas y controladores. Veamos el proceso de diseño por partes. Normalmente una aplicación web maneja datos almacenados en algún sitio, en este caso una base de datos, y permite crear, leer, actualizar y borrar dicha información. Es por ello que el diseño de la aplicación depende fuertemente del modelo de datos y suele seguirse el siguiente orden:

1. Diseñar y crear la base de datos.
2. Crear los modelos.
3. Crear los controladores.
4. Crear y ajustar las vistas.

La base de datos es el primer elemento en diseñar. La convención de nomenclatura de CakePHP especifica que las tablas deben estar nombradas en plural ya que almacenan varios elementos: la tabla *DATOS* almacena elementos de tipo *DATO*. Una vez la base de datos está lista, es necesario indicar cómo acceder a ella en el fichero *app/config/database.php* (en una instalación limpia de CakePHP probablemente se llame *app/config/database.php.default*). En él se pueden añadir varias bases de datos indicando el tipo de driver a utilizar, como puede ser *mysql* o *sqlite*, la máquina y el puerto dónde se aloja, el nombre de usuario y la contraseña, el nombre de la base de datos, etc.

Area	Type of Process	File Name and Location in Application
Model	Handles all database functions	app/models/{Model name}.php
View	Handles the presentation layer and displays, including Ajax output	app/views/{Controller name}/{View name}.ctp
Controller	Handles all logic and requests	app/controllers/{Controller name}_controller.php

Figura 40. Convenciones de nomenclatura en CakePHP [29].

El siguiente paso es crear el conjunto de modelos, vistas y controladores. Partiendo de que el nombre del modelo es el nombre de la tabla en singular, la Figura 40 presenta un resumen de la relación de los nombres de los distintos ficheros. Es posible que algunas de las tablas permitan organizar los datos en la base de datos pero no necesiten vistas ni/o un controlador; en cualquier caso siempre debe haber como mínimo un modelo ya que es lo que permite a CakePHP acceder a su información. Para una tabla de nombre *DATOS* la forma de nombrar los ficheros de los componentes es la siguiente:

- Modelo *dato.php*: es un fichero de nombre el de la tabla en singular con extensión *.php* que especifica la información necesaria para el tipo *DATO*, incluyendo las relaciones con otras tablas, y que se almacena en el directorio *app/models*. En este caso se maneja el nombre en singular ya que el modelo representa la interacción con una instancia de la tabla cada vez.
- Controlador *datos_controller.php*: es un fichero de nombre el de la tabla seguido de un guión bajo y la palabra *controller* con extensión *.php* que especifica toda la lógica a realizar en las peticiones acerca de los datos de tipo *DATO*. Se almacena en el directorio *app/controllers/*.
- Vistas: son tantos ficheros *.ctp* como vistas se requieran para los datos de tipo *DATO*, teniendo en cuenta que para cada vista existente debe haber una acción con el mismo nombre en el controlador. Estos ficheros combinan lenguajes como HTML, PHP o JAVASCRIPT para generar el contenido a enviar al cliente. Se almacenan en el directorio *app/views/{nombre_tabla}*. En nuestro ejemplo el directorio se llamaría *app/views/datos/*.

Además de modelos, controladores y vistas, CakePHP tiene otros elementos como son los componentes, los ayudantes, los elementos o los *layouts*. Los componentes contienen acciones o funciones que pueden ser utilizadas desde cualquier controlador, y se almacenan en el directorio *app/controllers/components*. Los ayudantes proporcionan funcionalidad a las vistas, simplificando el código que el programador necesita escribir por ejemplo para crear enlaces mediante el uso del ayudante HTML o para formularios mediante el ayudante FORM; se almacenan en *app/views/helpers*. Los elementos son los equivalentes a los componentes para las vistas; se pueden definir como piezas de código HTML que se repiten a lo largo de ellas por lo que es más cómodo y limpio mantenerlas en ficheros aparte en *app/views/elements*. Los layouts permiten definir diseños que se repiten en varias páginas, y se almacenan en *app/views/layouts*. Los comportamientos permiten añadir acciones complejas a la interacción con la base de datos; se almacenan en *app/models/behaviors*. Los *DataSources* son los encargados de interactuar con la base de datos. Los que vienen preinstalados en CakePHP interactúan con MySQL, PostgreSQL, Microsoft SQL Server 2000 o SQLite entre otros; se almacenan en *app/models/datasources*.

Anexo II: Certificados digitales X.509. Instalación de un servidor ILIP seguro mediante certificados digitales X.509 y OpenSSL.

Este anexo consta de varios apartados en los que se tratará primero el formato de los certificados digitales X.509 [39] y del fichero de configuración de la herramienta OpenSSL y por último las elecciones tomadas y los pasos realizados para crear un servidor web seguro dentro de la plataforma ILIP mediante el uso de una jerarquía de autoridades de certificación creadas con la herramienta OpenSSL.

Certificados X.509.

El formato de certificados X.509 es un estándar del ITU-T (*International Telecommunication Union-Telecommunication Standarization Sector*) y el ISO/IEC (*International Standards Organization / International Electrotechnical Commission*) **¡Error! No se encuentra el origen de la referencia.** cuya primera versión se publicó n 1988. En 1993 fue extendido para incluir dos nuevos campos opcionales que soportaran el control de acceso a directorios. En 1996 fue publicada la tercera versión, que permite la extensión con campos adicionales. Los elementos del formato de un certificado digital X.509 v3 [41] son:

- *Certificate* (certificado):
  - *Version* (versión): número de versión del certificado codificado (1, 2 o 3).
  - *Serial number* (número de serie): entero asignado por la autoridad certificadora. Cada certificado emitido por una CA debe tener un número de serie único.
  - *Algorithm ID* (identificador del algoritmo de firmado): algoritmo utilizado en la firma del certificado (como por ejemplo el RSA o el DSA).
  - *Issuer* (nombre del emisor): identifica la CA que ha emitido el certificado.
  - *Validity* (periodo de validez): periodo de tiempo durante el que el certificado es válido y la CA está obligada a mantener información sobre su estado. Consiste dos fechas: *Not Before* es la fecha en la que el certificado empieza a ser válido y *Not After* la fecha después de la cual el certificado deja de serlo.
  - *Subject* (nombre del sujeto): identidad cuya clave pública está certificada en el campo siguiente. El nombre debe ser único para cada entidad certificada por una CA dada, aunque puede emitir más de un certificado con el mismo nombre si es para la misma entidad.
  - *Subject Public Key Info* (información de clave pública del sujeto): consiste en dos elementos *Public Key Algorithm* es el identificador del algoritmo con el que se emplea la clave y *Subject Public Key* es la clave.
  - *Issuer Unique Identifier* (identificador único del emisor): campo opcional que permite reutilizar nombres de emisor.
  - *Subject Unique Identifier* (identificador único del sujeto): campo opcional que permite reutilizar nombres de sujeto.

- *Extensions* (extensiones).

- *Certificate Signature Algorithm* (algoritmo de firmado del certificado): algoritmo utilizado para firmar el certificado.
- *Certificate Signature* (firma del certificado): firma digital del certificado.

El nombre almacenado en los campos *issuer* y *subject* es lo que se denomina un *Distinguished Name*, es decir, el nombre que representa unívocamente a la entidad. Está compuesto jerárquicamente por atributos y sus correspondientes valores según el estándar X.500. Sin embargo, hay que tener en cuenta que las aplicaciones que procesan los certificados (tales como los navegadores) no son capaces de procesar cualquier atributo sino que se definió una recomendación [43] que especifica los atributos que sí se deben procesar. Los más típicos son:

- *commonName*
- *countryName*
- *emailAddress*
- *locality*
- *organization*
- *organizationalUnitName*
- *stateOrProvinceName*
- *surname*

Es posible utilizar otros siempre teniendo en cuenta que lo más probable es que sean ilegibles para las aplicaciones, por lo que el certificado será aceptado o no según la configuración de cada aplicación. Por ejemplo, en este proyecto se hace uso del atributo *userID*, que Apache no entiende, por lo que fue necesario realizar un procesamiento sobre el certificado dentro de la aplicación para poder acceder al valor de dicho campo.

Las extensiones del X.509 v3 proporcionan una manera de asociar información adicional a sujetos, claves públicas, etc. Un campo de extensión tiene tres partes:

1. *extnId* (tipo de extensión): proporciona la semántica y el tipo de información (cadena de texto, fecha u otra estructura de datos) para un valor de extensión.
2. *extnValue* (valor de la extensión): valor actual del campo.
3. *critical* (indicador de importancia): booleano que indica a una aplicación si es seguro ignorar el campo de extensión si no reconoce el tipo (valor por defecto *false*). Permite implementar aplicaciones que trabajan de modo seguro con certificados y que evolucionan según se añaden nuevas extensiones.

El ITU y el ISO/IEC han desarrollado y publicado un conjunto de extensiones estándar en un apéndice al X.509 v3, entre las que destacan las siguientes:

- *Basic constraints* (limitaciones básicas): indica si certificado es de una CA y el máximo nivel de profundidad de un camino de certificación a través de esa CA.
- *Certificate Policies* (política de certificación): contiene las condiciones bajo las que la CA emitió el certificado y el propósito del certificado.
- *CRL Distribution Points* (puntos de distribución de listas de revocación): indican cómo se obtiene la información sobre las listas de revocación.

- *Key Usage* (uso de la clave): restringe el propósito de la clave pública certificada, indicando, por ejemplo, que la clave sólo se debe usar para firmar, cifrar claves o datos, etc. Dado que la clave sólo está certificada para un propósito suele marcarse como importante.

Formato del fichero de configuración de la herramienta OpenSSL.

Dado que la implementación a realizar de la jerarquía es un prototipo simple apenas para generar peticiones de firma y firmarlas, sólo se van a utilizar unas pocas de todas las posibles opciones que ofrece la herramienta openssl:

- **-genrsa**: para generar los pares de claves pública-privada.
- **-req**: para generar peticiones de firma de certificados y certificados autofirmados.
- **-ca**: para firmar peticiones

La herramienta OpenSSL utiliza un fichero de configuración en el que se indica toda la información necesaria para la CA y los datos a incluir en los certificados y peticiones de firma: valores por defecto de los campos obligatorios y qué campos opcionales o qué extensiones añadir al certificado. Se divide en secciones, donde cada sección indica la configuración para una utilidad. A continuación se tratarán las secciones **ca** y **req** que almacenan la configuración para las opciones –ca y –req, respectivamente. La sección **ca** [43] indica la configuración para la generación de peticiones de firma de certificado. Los campos obligatorios son:

- *new_certs_dir*: directorio donde los nuevos certificados se almacenan.
- *certifícate*: fichero que contiene el certificado de la CA.
- *prívate_key*: fichero que contiene la clave privada de la CA.
- *default_md*: especifica el algoritmo de resumen de mensaje. Algunos valores son *md5*, *sha1* o *mdc2*, siendo *md5* el valor por defecto.
- *database*: fichero que almacena la base de datos textual.
- *serial:* fichero que almacena el siguiente número de serie a utilizar.

Entre los opcionales se destacan:

- *default_days*: número de días de validez del certificado.
- *x509_extensiones*: especifica el nombre de la sección que contiene las extensiones a añadir a un certificado. Si no aparece se crea un certificado v1 y si está presente aunque esté vacía se crea un certificado v3.
- *crl_extensions*: especifica el nombre de la sección que contiene las extensiones a añadir a las peticiones de revocación de certificados.
- *policy:* especifica el nombre de la sección que indica si los campos del certificado son obligatorios y deben coincidir o no con los del certificado de la CA. La sección consiste en una lista de variables correspondiente a los campos que conforman el campo *Subject*. Un valor *match* indica que la variable debe

valer lo mismo que en el certificado de la CA. Un valor *supplied* indica que es suficiente con que esté presente. Un valor *optional* indica que puede estar o no presente.

- *copy_extensions:* su ausencia o un valor *none* implica que las extensiones presentes en la petición son ignoradas y no se copian al certificado. Un valor de *copy* implica que cualquier extensión que no esté ya presente es copiada y un valor de *copyall* sobreescribe el valor de las extensiones que ya estén presentes. El principal uso de esta opción es permitir que las peticiones contengan extensiones como *subjectAltName*. Sin embargo, hay que valorar el riesgo de copiar extensiones que provengan del solicitante, ya que podría asignarse valores que no le correspondan, como por ejemplo, decir que es una CA cuando el certificado en realidad corresponde a un usuario final.

La sección **req** [44] indica la configuración para la generación de peticiones de firma de certificado. Entre las opciones disponibles se destacan:

- *default_md*: especifica el algoritmo de resumen de mensaje. Algunos valores son *md5*, *sha1* o *mdc2*, siendo *md5* el valor por defecto.
- *req_extensions*: especifica el nombre de la sección que contiene las extensiones a añadir a las peticiones de certificado.
- *x509_extensiones*: especifica el nombre de la sección que contiene las extensiones a añadir a un certificado autofirmado, es decir, cuando se utiliza la opción de línea de comandos *–x509*.
- *attributes*: especifica el nombre de la sección que contiene los atributos para las peticiones, como por ejemplo *challengePassword* o *unstructuredName*.
- *distinguished_name*: especifica el nombre de la sección que contiene los campos para formar el DN cuando se genera un certificado o petición de firma.

El formato de estas dos últimas secciones depende del valor de un campo denominado *prompt*, que indica si los valores del DN se han de pedir al usuario o tomarlos del fichero de configuración. Si su valor es *no*, entonces ambas secciones consisten en una lista de pares *nombre_campo=valor*:

CN=My Name
OU=My Organization
emailAddress=someone@somewhere.org

Si el campo *prompt* no está presente o su valor no es *no*, entonces las líneas de estas secciones tienen el formato:

fieldName = ”prompt”
fieldName_default = ”default field value”
fieldName_min = 2
fieldName_max = 5

donde *fieldName* toma el valor cualquier identificador de objeto como *commonName* o *CN*, *countryName, localitName, organizationName, organizationUnitName, stateOrProvinceName*o también *email Address, name, surname, givenName o dnQualifier*. La cadena de texto *“prompt”* indica al usuario datos sobre el campo. Si el

usuario no introduce ningún valor, entonces el valor por defecto es usado y si éste no existe, el campo no se incluye. Si se quiere omitir directamente el usuario puede introducir el carácter ".". Las otras dos directivas especifican los valores mínimo y máximo. Si algún campo puede tener varios valores, se especifica un número antes del nombre, como por ejemplo *0.organizationName*.

Las secciones que contienen las extensiones a añadir a las peticiones o los certificados consisten en un conjunto de líneas de la forma:

extensión_name=[critical,] extensión_options

La palabra *critical* indica si la extensión es o no crítica. Una extensión crítica no debe ser ignorada si no es posible leerla. El formato de *extensión_options* depende del valor de *extensión_name*. Los posibles tipos de extensiones son *string* (para *strings*), *multi-valued* (para extensiones con varios valores), *raw* y *arbitrary*. Un ejemplo del tipo *string* es:

nsComment="This is a Comment"

Para el tipo *multivalued* existen dos formatos. El formato corto es:

basicConstraints=critical,CA:true,pathlen:1

El format largo es:

basicConstraints=critical,@bs_section

[bs_section]
CA=true
pathlen=1

Entre las extensiones estándar soportadas por OpenSSL [45] se destacan:

- *basicConstraints:* es multievaluada. El primer valor indica si un certificado pertenece o no a una CA mediante *CA:true* o *CA:false,* respectivamente. Tras *CA:true* puede indicarse el número máximo de CAs que pueden aparecer bajo ésta mediante *pathlen*. El certificado de una CA debe usar obligatoriamente esta extensión.
- *KeyUsage:* es multievaluado e indica los usos permitidos de la clave: *digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, encipherOnly* y *decipherOnly.*
- *ExtendedKeyUsage:* es multievaluado e indica una lista de usos para la clave pública: *serverAuth, clienteAuth, codeSigning, ...*
- *subjetAltName:* permite incluir otros nombres que también identifiquen al sujeto del certificado mediante los campos *email, URI, DNS, RID, IP, dirName y otherName.* El campo *otherName* permite incluir cualquier tipo de datos asociados a un identificador de objeto.
- *issuerAltName:* igual que *subjAltName* pero para la CA que firma el certificado.
- *crlDistributionPoints:* es multievaludada e indica información referente a cómo obtener las listas de revocación de certificados.

Creación de una jerarquía de autoridades de certificación mediante OpenSSL.

En el apartado 3.2 Incrementando la seguridad del sistema: HTTPS y certificados digitales se han explicado las bases de HTTPS: criptografía asimétrica, certificados digitales y jerarquías de autoridades de certificación. A continuación se presentan desglosados y explicados para una mayor comprensión del proceso todos los pasos para crear la jerarquía de autoridades de certificación del sistema y cómo usarlas para generar los certificados para los distintos elementos. Sin embargo, todos estos pasos han sido automatizados en scripts.

En la Figura 41 se recuerda la jerarquía de CAs que se va a implementar: una CA raíz RootCA, una CA de usuarios UserCA y una CA de servidores ServerCA. Como ya se mencionó al tratar sobre el tema en el apartado 3.2, en un sistema real las CAs estarán instaladas en diferentes máquinas bajo fuertes requisitos de seguridad que prevengan que la jerarquía pueda verse comprometida pero para la implementación se ha elegido mantener la estructura de las tres CAs en la misma máquina bajo el directorio */var/ca*, ya que de momento sólo se dispone de dos máquinas para desplegar el sistema.

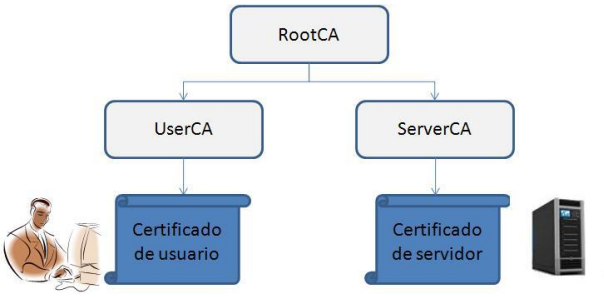


Figura 41. Jerarquía de autoridades de certificación a implementar.

El fichero de configuración por defecto de OpenSSL*/etc/ssl/openssl.conf* contiene la estructura de directorios, los campos obligatorios u opcionales y los valores por defecto para muchos de los elementos que se necesitan para la creación de autoridades de certificación, peticiones de firma, firma y revocación de certificados, entre otros. Si se desea utilizar otros nombres de directorio o modificar algunos parámetros será necesario bien modificar dicho fichero directamente o bien generar uno propio para la CA y especificar que se utilice ese en lugar del por defecto, que es el camino elegido. Los ficheros de configuración a utilizar se denominan *root-ca*.cnf, server-*ca.cnf* y *user-ca.cnf*. El fichero por defecto *openssl.conf* permite la creación de CAs bastante básicas pero con la suficiente funcionalidad para el propósito de este proyecto, por lo que los otros tres se han basado en él, con las diferencias que se explicaron en el apartado 3.2.

El comando openssl no genera ni la estructura de directorios ni ninguno de estos ficheros por lo que es necesario crearlos manualmente antes de utilizarlo. Una vez se tiene toda la estructura creada y los ficheros de configuración preparados, se puede comenzar a usar el comando openssl para generar cada una de las CAs. Los pasos para generar la CA raíz son:

1. Situarse en el directorio raíz */var/ca/root-ca/*.
2. Generar un par de claves pública y privada y proteger la privada mediante:

   openssl genrsa -aes256 -out private/root-ca_key.pem 4096

chmod 400 private/root-ca_key.pem

3. Generar y autofirmar el certificado:

   *openssl req -config root-ca.cnf -new -x509 -days 3650 *
   *-key private/root-ca_key.pem *
   -out certs/root-ca_cert.pem

Si se desea comprobar la clave se puede usar el comando:

openssl rsa -noout -text -in private/root-ca_key.pem

Si se desea comprobar los contenidos del certificado, se puede utilizar el comando:

openssl x509 -text -noout -in certs/root-ca_cert.pem

Los pasos para generar la ServerCA son:

1. Situarse en el directorio raíz */var/ca/server-ca*/.
2. Generar un par de claves pública y privada y proteger la privada:

   openssl genrsa -aes256 -out private/server-ca_key.pem 4096

   chmod 400 private/server-ca_key.pem

3. Generar una nueva petición de firma de certificado en *certs/* utilizando el fichero de configuración de RootCA:

   *openssl req –config ../root-ca/root-ca.cnf -new *
   *-key private/server-ca_key.pem *
   -out certs/server-ca_csr.pem

4. Enviar la petición a RootCA para que firme el certificado:

   cp certs/server-ca_csr.pem ../root-ca/csrs/

5. RootCA firma el certificado (puesto que están en la misma máquina no es necesario pasar a su directorio */var/ca/root-ca*/ pero hay que tener cuidado con las rutas) y lo envía a ServerCA:

   *openssl ca –config ../root-ca/root-ca.cnf *
   *-in ../root-ca/csrs/server-ca_csr.pem *
   -out ../root-ca/csrs/server-ca_cert.pem

   cp ../root-ca/csrs/server-ca_cert.pem certs/

6. ServerCA necesita también la cadena de certificados que han firmado su certificado, por lo que se le envía el certificado de RootCA y se genera el fichero con la cadena incluyendo el certificado del servidor:

   cp ../root-ca/certs/root-ca_cert.pem certs/

   cat certs/root-ca_cert.pem certs/server-ca_cert.pem >
   certs/server-ca_chain.pem

Los pasos para generar la UserCA son exactamente los mismos, situándose en el directorio raíz */var/ca/user-ca*/ y utilizando los nombres *user-ca_key.pem, user-ca_csr.pem, user-ca_cert.pem* y *user-ca_chain.pem.*

Para verificar los certificados desde */var/ca/* se puede utilizar el siguiente comando:

openssl verify –CAfile ca_file.pem cert_file.pem

donde *ca_file.pem* contiene el certificado de la CA con la que validar el certificado y *cert_file.pem* es el certificado o cadena de certificados a validar.

Una vez las CAs han sido creadas, se procede a generar un certificado para un servidor web y un certificado para un usuario. De nuevo el primer paso es crear la estructura de ficheros, en este caso bajo el directorio */etc/ssl/*, que es donde se almacenan los certificados en el sistema. Para cada elemento (usuario o servidor) los únicos directorios necesarios son *certs, crl y csrs*. Los datos para el servidor se almacenarán bajo el nombre *ilipserver* (INDECT Lawful Interception Platform Server) y los del usuario Raquel Aparicio bajo el nombre *raquel_a*. Los pasos para generar un certificado final de un servidor son:

1. Situarse en el directorio raíz *etc/ssl/ilipserver*/.
2. Generar un par de claves pública y privada y proteger la privada:

   openssl genrsa -out private/ilipserver_key.pem 2048

   chmod 400 private/ilipserver_key.pem
3. Generar una nueva petición de firma de certificado en *certs/* utilizando el fichero de configuración de ServerCA:

   *openssl req -config /var/ca/server-ca/server-ca.cnf -new *
   *-key private/ilipserver_key.pem *
   -out csrs/ilipserver_csr.pem
4. Enviar la petición a ServerCA para que firme el certificado:

   cp csrs/ ilipserver_csr.pem /var/ca/server-ca/csrs/
5. ServerCA firma el certificado (no es necesario pasar a su directorio */var/ca/root-ca*/ pero hay que tener cuidado con las rutas) y lo envía al directorio del servidor:

   *openssl ca -config /var/ca/server-ca/server-ca.cnf *
   *-in /var/ca/server-ca/csrs/ilipserver_csr.pem *
   -out /var/ca/server-ca/csrs/ilipserver_cert.pem

   cp /var/ca/server-ca/csrs/ilipserver_cert.pem certs/
6. El servidor necesita también la cadena de certificados que han firmado su certificado:

   cp /var/ca/server-ca/certs/root-ca_cert.pem certs/

   cp /var/ca/server-ca/certs/server-ca_cert.pem certs/

   cp /var/ca/server-ca/certs/server-ca_chain.pem certs/

   *cat certs/server-ca_chain.pem certs/ilipserver_cert.pem > *
   certs/ilipserver_chain.pem

Los pasos para generar un certificado final de un usuario son prácticamente iguales añadiendo uno al final para generar un fichero importable desde un navegador:

1. Situarse en el directorio raíz *etc/ssl/user-raquel_a*/.

2. Generar un par de claves pública y privada y proteger la privada:

   openssl genrsa -out private /user-raquel_a_key.pem 2048

   chmod 400 private/ user-raquel_a_key.pem

3. Generar una nueva petición de firma de certificado en *certs/* utilizando el fichero de configuración de UserCA:

   *openssl req -config /var/ca/user-ca/user-ca.cnf -new *
   *-key private/ user-raquel_a_key.pem *
   -out csrs/ user-raquel_a_csr.pem

4. Enviar la petición a UserCA para que firme el certificado:

   cp csrs/ user-raquel_a_csr.pem /var/ca/user -ca/csrs/

5. UserCA firma el certificado (no es necesario pasar a su directorio */var/ca/user-ca/* pero hay que tener cuidado con las rutas) y lo envía al directorio del servidor:

   *openssl ca -config /var/ca/user-ca/user-ca.cnf *
   *-in /var/ca/user-ca/csrs/ user-raquel_a_csr.pem *
   -out /var/ca/user-ca/csrs/ user-raquel_a_cert.pem

   cp /var/ca/user-ca/csrs/ user-raquel_a_cert.pem certs/

6. Obtener la cadena de certificados que han firmado el certificado:

   cp /var/ca/user-ca/certs/root-ca_cert.pem certs/

   cp /var/ca/user-ca/certs/user-ca_cert.pem certs/

   cp /var/ca/user-ca/certs/user-ca_chain.pem certs/

   *cat certs/user-ca_chain.pem certs/user-raquel_a_cert.pem > *
   certs/user-raquel_a_chain.pem

7. Obtener un fichero PKCS#12:

   *openssl pkcs12 -export -in certs/user-raquel_a_cert.pem *
   -inkey private/user-raquel_a_key.pem -out user-raquel_a.p12

Desde */etc/ssl/* se pueden realizar la verificación de los distintos certificados de la misma manera en que se realizó en el caso de los certificados de las CAs.

Uso de los certificados digitales de cliente. Dispositivos de seguridad.

Ahora que los certificados ya han sido generados y entregados a sus correspondientes partes, el siguiente paso es configurar el sistema para utilizarlos. Como ya se comentó, el usuario tiene dos opciones: importar su certificado digital en formato PKCS#12 directamente al navegador o almacenarlo en algún dispositivo o *token* de seguridad. En nuestro caso emplearemos tarjetas inteligentes de tipo normal y de tipo SIM, con sus correspondientes lectores/grabadores: uno normal y uno de tipo *token* USB. A continuación se explicarán los pasos necesarios para emplear este tipo de dispositivos pero antes es necesario comentar que el dispositivo debe ser conectado al ordenador y la tarjeta introducida en él antes de acceder a la página web para que el navegador lo reconozca antes del establecimiento de la conexión. En cualquier caso, si se configuró el

certificado con protección por contraseña, será necesario que la introduzca al menos una vez durante la importación del certificado. Cuando el usuario introduzca la dirección web de la aplicación y se comience el establecimiento de la conexión segura, el navegador pedirá al usuario que escoja el certificado digital a enviar de entre todos los que hayan sido firmados por alguna entidad de la lista de CAs que el servidor envía como aceptables.

El procedimiento es exactamente el mismo para ambos tipos de dispositivos. Una vez la tarjeta ha sido introducida en el correspondiente lector se pueden utilizar los siguientes comandos [35]:

- Obtener la huella de la tarjeta, conocido como ATR:

  opensc-tool --atr

- Borrar la tarjeta (sólo cuando ya haya sido escrito anteriormente): se borrarán todas las claves, PINs, certificados y cualquier otro dato almacenado en ficheros PKCS#15.

  pkcs15-init –E

- Formatear la tarjeta: crea una estructura de ficheros para el usuario *Usuario* de PIN XXXX y PUK (para desbloquearla) YYYYYY:

  pkcs15-init --create-pkcs15 --profile pkcs15+onepin --use-default-transport-key --pin XXXX --puk YYYYYY --label "Usuario"

- Cambiar el PIN

  pkcs15-tool --change-pin

- Importar un certificado a la tarjeta (auth-id es el identificador del PIN; empezando desde el 01):

  pkcs15-init --store-private-key ruta_a_certificado/certificado.p12 --format pkcs12 --auth-id YY –pin XXXX

- Mostrar la información de la tarjeta:

  pkcs15-tool --dump

Por lo tanto, para conseguir una tarjeta con el certificado de Raquel Aparicio almacenado en raíz *etc/ssl/user-raquel_a*, los pasos son:

1. Conectar el lector correspondiente e introducir la tarjeta en él.
2. Borrar la tarjeta, sólo si no está en blanco, es decir, si es la primera vez que se utiliza la tarjeta omitir este paso. En caso contrario el comando dará un error.

   pkcs15-init –E

3. Formatear la tarjeta para Raquel Aparicio, con PIN 0000 y PUK 111111.

   pkcs15-init --create-pkcs15 --profile pkcs15+onepin --use-default-transport-key --pin 0000 --puk 111111 --label "Raquel Aparicio"

4. Importar el certificado para el PIN que corresponda, en este caso el primero y único y, por lo tanto, el 01. Si no se indica un valor --*id* se generará uno automáticamente.

pkcs15-init --store-private-key etc/ssl/user-raquel_a /raquel_a.p12 --format pkcs12 --auth-id 01 –pin 0000

Y así con todos los usuarios y todas las tarjetas que se desee utilizar,

Una vez las tarjetas han sido grabadas, es necesario configurar el navegador para utilizarlas. En el caso de Firefox (http://www.gooze.eu/howto/iceweasel-firefox-smartcard-and-token-howto/configuring-the-navigator-to-use-smart-cards) sobre una distribución Linux hay que ir a las opciones de configuración, abrir la pestaña de *Avanzado* y dentro de ella *Cifrado*. Al pulsar sobre *Dispositivos de Seguridad* y dentro del mismo sobre *Cargar*, aparece una ventana en la que hay que introducir el nombre del módulo (*OpenSC*) y la ruta a su librería *(/usr/lib/opensc-pkcs11.so*). Tras guardar los cambios y reiniciar Firefox, si se visualizan los dispositivos de nuevo, debería aparecer la tarjeta. Al pulsar el botón *Iniciar Sesión*, se pedirá al usuario que introduzca el *PIN*, y el certificado pasará a estar disponible para su uso hasta que se retire la tarjeta del lector.

Uso de los certificados digitales de servidor. Instalación de un servidor web seguro.

La aplicación web ésta está implementada sobre el servidor web Apache2 que es un servidor HTTP de código abierto para varias plataformas, entre ellas Unix y Microsoft. Como ya se mencionó, para realizar todo el proceso de instalación existe un script que realiza los pasos que se incluyen a continuación:

- Preparar el sistema instalando todos los paquetes necesarios y configurando las variables necesarias, como por ejemplo, la variable de entorno de Axis para poder realizar llamadas a los *plugins* remotos.
- Preparar el servidor Apache y el servidor de contenidos VSFTP.
- Preparar la librería del módulo distribuidor de contenidos y generar su fichero de configuración con la dirección IP concreta de la máquina.
- Preparar los *plugins* locales.
- Preparar el módulo de traducción VoIP-texto, instalando previamente las herramientas necesarias: Sphinx, SphinxBase, SphinxTrain y PocketSphinx.
- Instalar la herramienta base Xplico, preparando previamente las herramientas necesarias, de momento únicamente Videosnarf.
- Preparar la interfaz gráfica de la plataforma.

Estos pasos consisten básicamente en la compilación de los distintos módulos y la preparación de sus estructuras de ficheros y ficheros de configuración. Mencionaremos los pasos necesarios para instalar y configurar Apache. La instalación se realiza mediante el comando:

apt-get install apache2

Este servidor es altamente configurable pero carece de interfaz gráfica para ello por lo que se maneja mediante ficheros y línea de comandos. Los ficheros de configuración se encuentran en el directorio */etc/apache2/:*

- *apache2.conf*: configuración del directorio raíz por defecto, logs, procesos, etc.
- *httpd.conf*: configuraciones de usuario.
- *ports.conf*: configuraciones de puertos.
- *mods-available/*: directorio de módulos disponibles.
- *mods-enabled/*: directorio de módulos habilitados.
- *sites-available/*: configuración de sitios disponibles.
- *sites-enabled/*: configuración de los sitios habilitados.

Además su arquitectura es muy modular. Consta de una sección *core* sobre la que se van activando otros módulos según se requiera. La plataforma ILIP requiere activar explícitamente los módulos *ssl* para comunicaciones seguras usando los protocolos SSL (*Secure Sockets Layer*) y TLS (*Transport Layer Security*), *php5* (para php) y rewrite (para reescribir URL). Se utiliza el comando *a2enmod nombre_modulo* mediante:

s*udo a2enmod php5 rewrite ssl*

Tras estos sencillos pasos, el servidor Apache ya está listo para su uso. Posteriormente, y una vez que la aplicación web está preparada en su correspondiente directorio, en este caso */opt/xplico/xi/*, es necesario modificar dos ficheros teniendo en cuenta que los datos de los certificados del servidor se encuentran bajo */etc/ssl/* y que la IP del servidor es X.X.X.X. El primer fichero es el */etc/apache2/ports.conf* de Apache, donde se indican los puertos sobre los que hay aplicaciones y por lo tanto el servidor debe escuchar. Anteriormente Xplico funcionaba sobre el puerto 9876 pero ahora que se utiliza el protocolo *https:* se emplea el puerto seguro 443, por lo que el fichero queda como se observa en la Figura 42.

```
NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    Listen 443
</IfModule>
```

Figura 42. Fichero de Apache de configuración de puertos /etc/apache2/ports.conf.

El segundo fichero es el de configuración del sitio web, */etc/apache2/sites-available/ilip* al que hay que añadir ciertas directivas que permiten al servidor encontrar la información necesaria sobre los certificados y claves:

- *SSL Engine on*: activa el motor SSL,
- *SSLRequireSSL*: no permite el acceso si no se utiliza el protocolo HTTPS,
- *SSLCertificateFile*: ruta al certificado del servidor,
- *SSLCertificateKeyFile*: ruta a la clave privada del servidor,
- *SSLCertificateChainFile*: ruta a la cadena de autoridades de certificación del certificado del servidor,
- *SSLCACertificateFile*: ruta a la cadena de autoridades de certificación para los certificados cliente,

- *SSLVerifyDepth 2*: máximo nivel de jerarquías de autoridades de certificación.

Aparte de las anteriores, también es necesario añadir las dos directivas siguientes, que serán útiles para el uso del servidor LDAP:

- *SSLVerifyClient optional*: indica que primero se pida el certificado de cliente pero que no es imprescindible que se presente uno para acceder al sitio web,
- *SSLOptions +StdEnvVars*: crea variables de entorno de SSL con los campos del certificado, accesibles desde PHP.

Por motivos prácticos, se han añadido también otras que realizan una redirección de HTTP a HTTPS, simplemente para que en lugar de devolver un error y que el usuario tenga que reescribir la URL mediante *https:*, esto se haga automáticamente. Para ello se utiliza las directivas *RewriteEngine on* y *RewriteRule*. El contenido final de este fichero se muestra en la Figura 43.

Una vez la aplicación está instalada y los ficheros de configuración han sido modificados ya sólo queda habilitar el sitio web y reiniciar Apache mediante:

sudo a2ensite ilip

sudo /etc/init.d/apache2 restart

Si se quisieran cargar ficheros pcap a través de la interfaz web sería necesario además modificar un tercer fichero */etc/php5/apache2/php.ini* cambiando las directivas *post_max_size* al valor de 800M y *upload_max_filesize* al valor de 400M. Esto permite que el servidor reciba ficheros de hasta estos tamaños.

```
<VirtualHost *:80>
        #allow URL rewriting
        RewriteEngine on
        #rewriting rute, everything from http, do it https
        #L means last rewriting rule, R means the requested page has moved
        RewriteRule ^(.*)$ https://X.X.X.X$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
    ServerAdmin info@xplico.org

    DocumentRoot /opt/xplico/xi
    <Directory "/opt/xplico/xi">
        Options All
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>

    ErrorLog /var/log/apache2/xplico_error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel error

    CustomLog /var/log/apache2/xplico_access.log combined

        #OpenSSL
    #Certificates: activate SSL engine and specify where the certificate and key are
    SSLEngine on
    #server PEM-encoded X.509 Certificate file
    SSLCertificateFile /etc/ssl/ilip_certificates/ilipserver/certs/ilipserver_cert.pem
    #server PEM-encoded private key file
    SSLCertificateKeyFile /etc/ssl/ilip_certificates/ilipserver/private/ilipserver_key.pem
    #file of PEM-encoded Server CA Certificates
    SSLCertificateChainFile /etc/ssl/ilip_certificates/ilipserver/certs/server-
ca_chain_openssl.pem

    #file of concatenated PEM-encoded CA Certificates for client authorization
    SSLCACertificateFile /etc/ssl/ilip_certificates/user-ca_data/user-ca_chain_openssl.pem
    #maximum depth of CA certificates in client certificate verification
    SSLVerifyDepth 2
    #type of client certificate verification
    SSLVerifyClient optional

    #configure various SSL engine run-time options: create SSL related environment variables
    SSLOptions +StdEnvVars +ExportCertData
    #not really needed, gives access to the actual certificate: +ExportCertData

        <directory />
                #deny access when SSL is not used for the request
                SSLRequireSSL
        </directory>
</VirtualHost>
```

Figura 43. Fichero de Apache de configuración del sitio web /etc/apache2/sites-available/ilip.

Anexo III: LDAP y LDIF

LDAP (Lightweight Directory Access Protocol).

LDAP es un protocolo de nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuído para buscar información en un entorno de red. Un directorio es un conjunto de objetos con atributos organizados en una manera lógica y jerárquica. En el caso particular de LDAP normalmente se almacena la información de autenticación de los usuarios de un sistema, aunque puede almacenar cualquier otro tipo de datos. Esta definición puede ser difícil de entender si no se está familiarizado con el tema por lo que vamos a entrar un poco más en detalle. La información a continuación se ha obtenido de la página web oficial de Ubuntu [35].

- Un directorio es un árbol de entradas de directorio.
- Una entrada consta de un conjunto de atributos.
- Cada entrada tiene un identificador único denominado DN (*Distinguished Name*) que se construye anteponiendo al DN de la entrada padre el RDN (*Relative Distinguished Name*) de la entrada hijo, formada por algunos de sus atributos.
- Una entrada puede pertenecer a más de una clase de objetos. Esto permite definirlas mediante atributos de cada una de las clases de objetos a las que pertenece.
- Una clase de objeto define la colección de atributos que pueden usarse para definir una entrada. Estos pueden ser atributos-debe (cada instancia debe incluirlos) o atributos-puede (pueden definirse u omitirse al crear el objeto).
- Un atributo tiene un nombre y uno o más valores y se define en esquemas.
- Un esquema define las propiedades y sus valores asociados de cada elemento del árbol LDAP. Esto incluye las clases de objeto y sus atributos, entre otros.

Los atributos y las clases de objeto se estandarizan y registran en el IANA, lo que permite reutilizarlos. Algunos de los tipos básicos para las clases de objetos son:

- grupos en el directorio, que forman listas no ordenadas de objetos individuales o de grupos de objetos.
- emplazamientos, como por ejemplo el nombre del país y su descripción.
- organizaciones que están en el directorio.
- personas que están en el directorio.

Las principales diferencias de un servicio de directorio respecto a una base de datos relacional son primero la organización jerárquica, con información más descriptiva y basada en atributos, y segundo que su principal objetivo son las búsquedas eficientes ya que la lectura de información se produce con mayor frecuencia que la escritura.

El servicio de directorio LDAP se basa en un modelo cliente-servidor, donde uno o más servidores LDAP contienen los datos que forman el árbol del directorio. Si varios servidores contienen la misma información se denominan réplicas. El otro caso es que

cada servidor aloje simplemente un subárbol comenzando por una entrada específica y sus hijos. En ese caso puede almacenar referencias a otros servidores para poder devolver la referencia al servidor que aloja esa parte del árbol de directorio o puede soportar el encadenamiento, es decir, el propio servidor contacta al otro servidor y devuelve el resultado al cliente.

Un cliente inicia una sesión de LDAP conectándose a un servidor LDAP; el puerto por defecto es el TCP 389. Una vez iniciada el cliente manda peticiones al servidor y este le envía las respuestas. Las operaciones que el cliente puede realizar son:

- *Start TLS* para iniciar una conexión segura.
- *Bind* para autentificarse y opcionalmente especificar una versión LDAP.
- *Search* para buscar y obtener entradas de directorio.
- *Compare* para probar si una entrada contiene un determinado valor de atributo.
- *Add* para añadir una nueva entrada.
- *Delete* para borrar una entrada.
- *Modify* para modificar una entrada.
- *Modify Distinguished Name* para modificar o renombrar una entrada.
- *Abandon* para abortar una petición previa.
- *Extended Operation* para definir una operación.
- *Unbind* para cerrar la conexión.

El formato URL de LDAP consta de los siguientes componentes:

ldap://servidor:puerto/DN?attributos?ámbito?filtro?extensiones

- servidor LDAP donde se realiza la consulta (FQDN o IP),
- puerto de red del servidor LDAP,
- DN que sirve de base de la búsqueda,
- lista de atributos, separada con comas, a devolver,
- ámbito de búsqueda: *base* (por defecto), *one* o *sub*,
- filtro de búsqueda,
- extensiones al formato URL de LDAP.

Para usar los valores por defecto simplemente se omite el valor del elemento y los caracteres especiales deben ser codificados con signos de porcentaje. Para utilizar SSL existen dos opciones. La primera es usar LDAP sobre SSL mediante el esquema *ldaps:* (sobre el puerto 636) similar al de *ldap:*, aunque no está estandarizado. La segunda consiste en emplear LDAP con SLL mediante el esquema normal *ldap:*y la operación *StartTLS*.

La entrada de más alto nivel del directorio normalmente basa su DN en el nombre de dominio del servidor LDAP. Por ejemplo, en nuestro caso la organización tiene el nombre de dominio *indect.eu* y el servidor se denomina *ldapserver.indect.eu*, por lo que

el DN de la entrada raíz será *dc=indect,dc=eu*. En este caso, la URL para acceder al nivel más alto de la organización sería:

ldap://ldapserver.indect.eu/dc=indect,dc=eu

Para obtener todos los atributos de la entrada "Raquel Aparicio" en este servidor usaríamos:

ldap:// ldapserver.indect.eu/cn=Raquel%20Aparicio,dc=indect,dc=eu

donde %20 representa el espacio. También se podría utilizar un filtro para buscar el atributo *givenName* y buscar en el servidor por defecto:

ldap:///dc=indect,dc=eu??sub?(givenName=Raquel)

LDIF (LDAP Interchange Format).

Ahora que ya sabemos cómo comunicarnos con un servidor LDAP desde un cliente, veamos cómo administrar el servidor mediante ficheros LDIF. Los ficheros LDIF se utilizan para construir la estructura del árbol de directorio, añadir entradas o modificarlas e importar o exportar el directorio. Se trata de ficheros de texto por lo que se pueden crear y editar mediante cualquier editor de texto. A continuación se presenta un pequeño resumen básico de LDIF obtenido de la página web de Zytrax [37].

Los ficheros LDIF poseen cinco tipos de líneas:

- Línea de directiva: empieza en la primera columna con un carácter que no sea ni espacio ni #.
- Línea de continuación: empieza en la primera columna con un espacio e indica que todos los caracteres a continuación son parte de la línea anterior.
- Línea en blanco: no hay ningún carácter (tecla de intro).
- Línea de comentario: empieza en la primera columna con una #.
- Línea de separación: empieza en la primera columna con un – e indica el fin de una secuencia de operador

Dentro de las líneas de directiva encontramos dos tipos de secuencias LDIF:

- Una secuencia de entrada es un grupo de directivas que típicamente comienzan con una directiva *dn:* y se aplican a la misma entrada. Empiezan y terminan con una línea en blanco.
- Una secuencia de operador es un grupo de directivas agrupadas bajo una directiva *changetype: modify* que se aplican a una única operación *add*, *delete* o *replace*.

Por último, veremos todas las posibles directivas del formato LDIF. Primero encontramos:

- *dn: DN* - define el DN sobre el que se van a aplicar las operaciones especificadas en las directivas posteriores. Debe precederse de una línea en blanco, excepto la primera entrada del fichero LDIF.

- *changetype: add/delete/modify/modrdn* - sigue a una directiva *dn* y define la operación a realizar sobre la misma: crear una entrada nueva (por defecto), borrar o modificar la entrada indicada, o modificar el *rdn* de la entrada (siempre y cuando esta no tenga hijos).

Siguiendo a una directiva *change type: add* podemos usar las siguientes directivas:

- *add: attributename* – irá seguida de la directiva *attributename: value*
- *objectclass: objectclassname* – indica que la entrada es de tipo *objectclassname;* irá seguida de la directiva *add: attributename* para especificar al menos el valor de los atributos-debe.

Siguiendo a una directiva *change type: modify* podemos usar las siguientes directivas:

- *add: attributename* – irá seguida de la directiva *attributename: value*
- *delete: attributename* – puede ir seguida de la directiva *attributename: value* para especificar qué atributo borrar; si no, todos los atributos de nombre *attributename* serán borrados
- *replace: attributename* – modifica todos los valores de *attributename* dejando sólo los especificados a continuación mediante la directiva *attributename: value*; si sólo se desea modificar un valor, es más cómodo utilizar *delete* seguido de *add*
- *objectclass: objectclassname* – funciona igual que dentro de la directiva *change type: add*

Siguiendo a una directiva *change type: modrdn* (*moddn* es un alias con la misma funcionalidad) podemos usar las siguientes directivas:

- *newrdn: RDN* – crea una copia de la entrada indicada dándole la RDN especificada; puede ir seguida de la directiva deleteoldrdn
- *deleteoldrdn: 0/1* – 0 mantiene la entrada original y 1 la borra; puede ir seguida de la directiva *newsuperior*
- *newsuperior: DN* – mueve la nueva entrada dentro del árbol

En el siguiente apartado se pueden ver algunos ejemplos para insertar usuarios en el sistema, para modificarlos y para eliminarlos.

Instalación de un servidor OpenLDAP

La información en Internet para instalar OpenLDAP puede ser un poco confusa debido a que se produjo un cambio en la forma de configurarlo. Anteriormente se utilizaba un único fichero de configuración *slapd.conf* y cuando había que hacer modificaciones, era necesario parar y arrancar de nuevo el servidor. Actualmente se prefiere el método denominado *cn=config*, donde la configuración se almacena también como un árbol de directorios aparte, lo que permite realizar modificaciones sin necesidad de parar y arrancar el sistema. En este anexo se explican los pasos básicos que se han seguido para esta aplicación, sacados de la página oficial de Ubuntu [35].

Durante la instalación de *slapd* se crea una configuración básica sobre la que trabajar: una base de datos cuyo DN raíz se basa en el nombre de dominio del servidor que incluye los esquemas clásicos *core*, *cosine*, *nis* e *inetorgperson*. Si se desea un DN raíz diferente, primero hay que editar el fichero */etc/hosts* y cambiar el nombre de dominio por el deseado. En nuestro caso el nombre del servidor es *ldapserver* y el nombre de dominio es *dc=indect,dc=eu*, por lo que hay que modificar dicho fichero escribiendo:

127.0.1.1 ldapserver.indect.eu ldapserver

Realizamos la instalación mediante el comando

apt-get install slapd ldap-utils

El proceso solicitará datos para el rootDN de la base de datos, que por defecto es *cn=admin,dc=indect,dc=eu.* Una vez terminada la instalación ya se pueden deshacer los cambios en el fichero */etc/hosts* y empezar a configurar la base de datos creada. Tras instalar el paquete, se han creado dos árboles: uno que almacena la configuración de slapd y otro que almacena el directorio propiamente dicho. El árbol que almacena la configuración de slapd se puede ver en forma de ficheros *.ldif* bajo */etc/ldap/slapd.d:*

```
/etc/ldap/slapd.d/
├── cn=config
│   ├── cn=module{0}.ldif
│   ├── cn=schema
│   │   ├── cn={0}core.ldif
│   │   ├── cn={1}cosine.ldif
│   │   ├── cn={2}nis.ldif
│   │   └── cn={3}inetorgperson.ldif
│   ├── cn=schema.ldif
│   ├── olcBackend={0}hdb.ldif
│   ├── olcDatabase={0}config.ldif
│   ├── olcDatabase={-1}frontend.ldif
│   └── olcDatabase={1}hdb.ldif
└── cn=config.ldif
```

No se deben hacer cambios directos sobre estos ficheros sino que se deben utilizar las herramientas que se acaban de instalar con el paquete. Por ejemplo, para visualizar este árbol se usa la herramienta *ldapsearch* de la siguiente manera:

sudo ldapsearch -Q -LLL -Y EXTERNAL -H ldapi:/// -b cn=config dn

El resultado del comando anterior es:

```
dn: cn=config

dn: cn=module{0},cn=config

dn: cn=schema,cn=config

dn: cn={0}core,cn=schema,cn=config

dn: cn={1}cosine,cn=schema,cn=config
```

dn: cn={2}nis,cn=schema,cn=config

dn: cn={3}inetorgperson,cn=schema,cn=config

dn: olcBackend={0}hdb,cn=config

dn: olcDatabase={-1}frontend,cn=config

dn: olcDatabase={0}config,cn=config

dn: olcDatabase={1}hdb,cn=config

Veamos el significado de algunas de estas entradas [38]:

1. Configuración general: *dn: cn=config*: *dn.* Bajo esta entrada se encuentran las directivas que se aplican al servidor en general, la mayoría de ellas orientadas al sistema o a la conexión.
2. Módulos: *dn: cn=module{0},cn=config*. Esta entrada especifica los conjuntos de módulos cargados.
3. Esquemas: *cn=schema,cn=config*. Esta entrada especifica los esquemas:
    - *dn: cn={0}core,cn=schema,cn=config*: esquema base o *core*
    - *dn: cn={1}cosine,cn=schema,cn=config*: esquema *cosine*
    - *dn: cn={2}nis,cn=schema,cn=config*: esquema *nis*
    - *dn: cn={3}inetorgperson,cn=schema,cn=config*: esquema *inetorgperson*
4. *Backend: dn: olcBackend={0}hdb,cn=config*. Las directivas bajo esta entrada se aplican a todas las bases de datos del tipo correspondiente.
5. Base de datos:
    - *dn: olcDatabase={-1}frontend,cn=config*. Base de datos que almacena las opciones a aplicar a las demás base de datos. Creada por el sistema.
    - *dn: olcDatabase={0}config,cn=config*. Base de datos que almacena la configuración. Creada por el sistema.
    - *dn: olcDatabase={1}hdb,cn=config.* Base de datos creada para ser utilizada por los usuarios, de tipo hdb.

El árbol correspondiente a esta última base de datos se visualiza mediante el comando:

ldapsearch -x -LLL -H ldap:/// -b dc=indect,dc=eu dn

cuyo resultado es

dn: dc=indect,dc=eu

dn: cn=admin,dc=indect,dc=eu

La primera entrada es la base del árbol y la segunda corresponde al administrador.

Ahora que el servidor LDAP está en funcionamiento, lo primero que hay que hacer es insertar el nuevo esquema, cuya descripción se muestra en la . Para ello es necesario crear el fichero *.ldif* de la Figura y ejecutar el comando:

sudo ldapadd -Y EXTERNAL -H ldapi:/// -f add_indect_schema.ldif

Para poblar la base de datos es necesario crear otro fichero LDIF con la información a añadir y ejecutar de nuevo el comando *ldapadd* para que lo procese. La información que queremos añadir a la base de datos son las aplicaciones del sistema y los usuarios y sus permisos para cada aplicación. Utilizaremos los ficheros de las Figuras 44 y 45, que insertaremos mediante el comando:

ldapadd -cx -D cn=admin,dc=indect,dc=eu -W -f nombre_fichero.ldif

```
dn: cn=indect,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: indect
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.1 NAME 'aid' DESC 'Application identifier' EQUALITY
caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.2 NAME 'userType' DESC 'User type' EQUALITY
caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.3 NAME 'create' DESC 'Create permission' EQUALITY
booleanMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.4 NAME 'read' DESC 'Read permission' EQUALITY
booleanMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.5 NAME 'update' DESC 'Update permission' EQUALITY
booleanMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.6 NAME 'delete' DESC 'Delete permission' EQUALITY
booleanMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.7 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.7 NAME 'permissionType' DESC 'Type of permission
object' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX
1.3.6.1.4.1.1466.115.121.1.15 )
olcObjectClasses: ( 1.3.6.1.4.1.4203.666.8 NAME 'application' DESC 'INDECT application data' SUP
top STRUCTURAL MUST ( aid $ permissionType $ name $ mail ) MAY ( description) )
olcObjectClasses: ( 1.3.6.1.4.1.4203.666.9 NAME 'appPermission' DESC 'Especifies that an user has
access to application aid' SUP top STRUCTURAL MUST ( aid ) )
olcObjectClasses: ( 1.3.6.1.4.1.4203.666.10 NAME 'ilipAppPermission' DESC 'Especifies special
permissions in ILIP applications' SUP appPermission STRUCTURAL MUST ( userType ) )
olcObjectClasses: ( 1.3.6.1.4.1.4203.666.11 NAME 'dbAppPermission' DESC 'Especifies special
permissions in DB applications' SUP appPermission STRUCTURAL MUST ( create $ read $ update $
delete ) )
```

Figura 44. Fichero LDIF correspondiente al esquema INDECT de las Figuras 25 y 26.

```
dn: ou=users,dc=indect,dc=eu
objectclass: organizationalUnit
ou: users

dn: ou=applications,dc=indect,dc=eu
objectclass: organizationalUnit
ou: applications
```

Figura 45. Fichero LDIF para crear la estructura del directorio.

```
#applications
dn: aid=portal,ou=applications,dc=indect,dc=eu
objectClass: application
aid: portal
permissionType: appPermission
name: INDECT portal
description: Platform which analyses interception contents
mail: indect_portal_admin@indect.eu

dn: aid=ilip,ou=applications,dc=indect,dc=eu
objectClass: application
aid: ilip
permissionType: ilipAppPermission
name: INDECT Lawful Interception Platform
description: Platform which analyses interception contents
mail: ilip_admin@indect.eu

dn: aid=db1,ou=applications,dc=indect,dc=eu
objectClass: application
aid: db1
permissionType: dbAppPermission
name: Database Application number 1
description: Application based on a database
mail: indect_db1_admin@indect.eu
```

Figura 46. Fichero LDIF para añadir las aplicaciones al directorio.

```
#raquel_a_admin
dn: uid=raquel_a_admin,ou=users,dc=indect,dc=eu
objectClass: inetOrgperson
cn: Raquel
sn: Aparicio
uid: raquel_a_admin
userPassword: ******
mail: pki_raquel_a_admin@example.org

#raquel_a_admin's permissions
dn: aid=portal,uid=raquel_a_admin,ou=users,dc=indect,dc=eu
objectClass: appPermission
aid: portal

dn: aid=ilip,uid=raquel_a_admin,ou=users,dc=indect,dc=eu
objectClass: ilipAppPermission
aid: ilip
userType: admin

#raquel_a
dn: uid=raquel_a,ou=users,dc=indect,dc=eu
objectClass: inetOrgperson
cn: Raquel
sn: Aparicio
uid: raquel_a
userPassword: ******
mail: pki_raquel_a@example.org

#raquel_a's permissions
dn: aid=portal,uid=raquel_a,ou=users,dc=indect,dc=eu
objectClass: appPermission
aid: portal

dn: aid=ilip,uid=raquel_a,ou=users,dc=indect,dc=eu
objectClass: appPermission
aid: ilip

#raquel_m
dn: uid=raquel_m,ou=users,dc=indect,dc=eu
objectClass: inetOrgperson
cn: Raquel
sn: Morenilla
uid: raquel_m
userPassword: ******
mail: pki_raquel_m@example.org

#raquel_m's permissions
dn: aid=portal,uid=raquel_m,ou=users,dc=indect,dc=eu
objectClass: appPermission
aid: portal

dn: aid=db1,uid=raquel_m,ou=users,dc=indect,dc=eu
objectClass: dbAppPermission
aid: db1
create: FALSE
read: TRUE
update: TRUE
delete: FALSE
```

Figura 47. Fichero LDIF para añadir los usuarios al directorio.

Anexo IV: Ejemplos de *plugins*.

En este anexo simplemente se va a incluir el código de algunos *plugins* locales y remotos, que utilizan las herramientas que se ponen a disposición de los desarrolladores.

- Plugin local HelloWorld desarrollado en C, Figuras 48 y 49.
- Plugin local HelloWorld desarrollado en *bash script*, Figuras 50, 51 y 52.
- Plugin local y remoto HelloWorld desarrollado en Java: en un mismo código tiene en cuenta la posibilidad de ser ejecutado en remoto (necesita descargar el contenido) o en local (el contenido ya es disponible. Las herramientas son capaces de generar un plugin remoto y otro local automáticamente. El funcionamiento consiste en descargar un contenido y devolver relevancia 1 con un mensaje de *HelloWorld*, a no ser que no sea posible descargar el contenido en cuyo caso devuelve error (-1) y el mensaje de error. Su código se incluye en la Figuras 53 y 54.

```
#include "local_plugin.h"

#include <stdio.h> //printf
#include <unistd.h> //sleep
#include <string.h> //strlen

#define PLUGIN_NAME "hello_world_plugin"

void pluginAnalyze(local_plugin_rqst_t* request, local_plugin_rsp_t* response);
void pluginTrain(local_plugin_rqst_t* request, local_plugin_rsp_t* response);
int main ( int argc, char * argv[] ){
  local_plugin_rqst_t request;
  int err = local_plugin_parse_args(argc, argv, &request);
  if (err != 0) {
    return PRIORITY_ERROR;
  }

  local_plugin_rsp_t response;
  //default priority
  response.process_relevance = PRIORITY_UNKNOWN;
  //default message
  response.process_message = "";

  //define your processing here: if you want to call a script, you may prefer using the
template for bash scripts
  if(request.request_command == ANALYSIS_COMMAND)
      pluginAnalyze(&request, &response);
  else
      pluginTrain(&request, &response);

  printf("[%s] My process message is: %s", PLUGIN_NAME, response.process_message);
 return response.process_relevance;
}
```

Figura 48.Código del *plugin* HelloWorld desarrollado en C (I)

```
void pluginAnalyze(local_plugin_rqst_t* request, local_plugin_rsp_t* response){
   response->process_message = "Hello World";
   response->process_relevance = 0;
}

void pluginTrain(local_plugin_rqst_t* request, local_plugin_rsp_t* response){
   char message [strlen("Training  with message ")+strlen(request->content_uri)
+strlen(request->process_message)];
   sprintf(message, "Training %s with message %s", request->content_uri, request-
>process_message);
   response->process_message = strdup(message);
   response->process_relevance = request->process_relevance ;
}
```

Figura 49.Código del *plugin* HelloWorld desarrollado en C (II)

```
#!/bin/bash

PRIORITY_5=5
PRIORITY_4=4
PRIORITY_3=3
PRIORITY_2=2
PRIORITY_1=1
PRIORITY_UNKNOWN=0
PRIORITY_ERROR=255
MAX_PRIORITY=$PRIORITY_5
MIN_PRIORITY=$PRIORITY_UNKNOWN
ANALYSIS_COMMAND_STR="analysis"
TRAINING_COMMAND_STR="training"
ANALYSIS_COMMAND_ARGC=9
TRAINING_COMMAND_ARGC=11
#script name
MYSELF=`basename $0`
REQUEST_COMMAND=""
CASE_CATEGORY=""
CASE_ID=""
CONTENT_NAME=""
CONTENT_TIMESTAMP=""
CONTENT_TYPE=""
CONTENT_URI=""
CONTENT_HASH=""
CONTENT_SIZE=""
PROCESS_RELEVANCE=""
PROCESS_MESSAGE=""

function local_plugin_print_help{
    echo "Usage: $MYSELF
{'$ANALYSIS_COMMAND_STR'|'$TRAINING_COMMAND_STR'} <category> <case>
<uri> <name> <time> <type> <hash> <size>  [<relevance> <message>]"
    echo
```

Figura 50. Código del *plugin* HelloWorld desarrollado en *bash script* (I)

```
    echo "        <category> : Category of the current case."
    echo "            <case> : Identifier of the current case."
    echo "             <uri> : URI to fecth the content from."
    echo "            <name> : Filename of the content."
    echo "            <time> : Timestamp when the content was captured."
    echo "            <type> : MIME type of the content."
    echo "            <hash> : MD5 hash of the content."
    echo "            <size> : Size of the content file in bytes."
    echo "       <relevance> : Relevance of the content.\n"
    echo "                     Only for '$TRAINING_COMMAND_STR' command."
    echo "         <message> : Message to be returned after processing"
    echo "                     Only for '$TRAINING_COMMAND_STR' command."
    echo
}

#$1 message (optional) $2 value (optional)
function local_plugin_print_error{
  if [ $# -eq 1 ]; then
     echo "$MYSELF: $1"
  elif [ $# -eq 2 ]; then
     echo "$MYSELF: $1: $2"
  else
     echo "$MYSELF: Unknown error"
  fi
  exit $PRIORITY_ERROR
}

function local_plugin_parse_args{
   REQUEST_COMMAND=$1
   if [ "$REQUEST_COMMAND" != "$ANALYSIS_COMMAND_STR" ] && [
"$REQUEST_COMMAND" != "$TRAINING_COMMAND_STR" ]; then
        local_plugin_print_help
        local_plugin_print_error "Unknown command" $REQUEST_COMMAND
   fi

   if [ "$REQUEST_COMMAND" == "$ANALYSIS_COMMAND_STR" ] && [ $# -ne
$ANALYSIS_COMMAND_ARGC ]; then
        local_plugin_print_help
        local_plugin_print_error "Invalid number of 'analysis' command arguments"
   fi

   if [ "$REQUEST_COMMAND" == "$TRAINING_COMMAND_STR" ] && [ $# -ne
$TRAINING_COMMAND_ARGC ]; then
        local_plugin_print_help
        local_plugin_print_error "Invalid number of 'training' command arguments"
   fi

   CASE_CATEGORY=$2
   CASE_ID=$3
   CONTENT_URI=$4
   CONTENT_NAME=$5
```

Figura 51. Código del *plugin* HelloWorld desarrollado en *bash script* (II)

```
    CONTENT_TIMESTAMP=$6
    case $CONTENT_TIMESTAMP in
        "|*[!0-9]*) local_plugin_print_error "Invalid <timestamp> argument"
$CONTENT_TIMESTAMP ;;
        *) ;;
    esac

    CONTENT_TYPE=$7
    CONTENT_HASH=$8
    CONTENT_SIZE=$9
    case $CONTENT_SIZE in
        "|*[!0-9]*) local_plugin_print_error "Invalid <size> argument" $CONTENT_SIZE ;;
        *) ;;
    esac
    if [ $CONTENT_SIZE -le 0 ] ; then
        local_plugin_print_error "Invalid <size> argument" $CONTENT_SIZE
    fi

    if [ "$REQUEST_COMMAND" == "$TRAINING_COMMAND_STR" ]; then
      PROCESS_RELEVANCE=${10}
      case $PROCESS_RELEVANCE in
        "|*[!0-9]*) local_plugin_print_error "Invalid <priority> argument"
$PROCESS_RELEVANCE ;;
        *) ;;
      esac
      if [ $PROCESS_RELEVANCE -lt $MIN_PRIORITY ] || [ $PROCESS_RELEVANCE -
gt $MAX_PRIORITY ]; then
        local_plugin_print_error "Invalid <priority> argument" $PROCESS_RELEVANCE
      fi

      PROCESS_MESSAGE=${11}
    fi
}

#variables are global, no need for arguments
function pluginAnalyze{
    echo "[hello_world_plugin] Hello World"
    exit 0
}

#variables are global, no need for arguments
function pluginTrain{
    echo "[hello_world_plugin] Training '$CONTENT_NAME' in '$CONTENT_URI' with
message '$PROCESS_MESSAGE'"
    exit $PROCESS_RELEVANCE
}

local_plugin_parse_args $@
if [ "$REQUEST_COMMAND" == "$ANALYSIS_COMMAND_STR" ]; then
    pluginAnalyze
else
    pluginTrain
fi
```

Figura 52. Código del *plugin* HelloWorld desarrollado en *bash script* (III)

```
package hw;

//structures to use, interface to implement and useful classes
import ilip.plugin.*;
import java.util.Properties;

public class HelloWorld implements IlipPluginInterface
{
	//plugin name: from manifest in jar file
	private String plugin_name = "hello_world_plugin";
	//properties to use
	private Properties prop = null;
	//message to return
	private String return_message = "[Hello World] ";

	public HelloWorld(){}
	public String getPluginName() { return this.plugin_name; }
	public void initializePlugin(Properties prop){ this.prop = prop; }
	public void destroyPlugin(){}

	public final Response analyzeRequest (Request analysisReq) {
		int return_relevance = 0;
		String message = "";
		String content_file = "/tmp/" + analysisReq.getContentHash();
		boolean content_available = false;

		if(IlipPluginUtils.isLocalFile(analysisReq.getContentURI())){
			content_available = true;
			content_file = analysisReq.getContentURI().toString();
		}
		else if( (content_available =
IlipPluginUtils.downloadFile(analysisReq.getContentURI(),
prop.getProperty("pluginFtpUser"), prop.getProperty("pluginFtpPwd"), content_file)) ==
false)
		{
			message = "Content could not be downloaded";
			return_relevance = -1;
		}

		if(content_available)
			message = analysisReq.getContentHash();

		Response res = new Response(analysisReq, return_relevance,
return_message + message);
		return res;
	}
```

Figura 53. Código del *plugin* HelloWorld desarrollado en JAVA (I)

```
        public final Response trainRequest (Request trainingReq) {
                int return_relevance = 0;
                String message = "";
                String content_file = "/tmp/" + trainingReq.getContentHash();
                boolean content_available = false;

                if(IlipPluginUtils.isLocalFile(trainingReq.getContentURI())){
                        content_available = true;
                        content_file = trainingReq.getContentURI().toString();
                }
                else if( (content_available =
IlipPluginUtils.downloadFile(trainingReq.getContentURI(),
prop.getProperty("pluginFtpUser"), prop.getProperty("pluginFtpPwd"), content_file)) ==
false)
                {
                        message = "Content could not be downloaded";
                        return_relevance = -1;
                }

                if(content_available)
                        message = "Training " + trainingReq.getContentName() +" in "+
trainingReq.getContentURI() +" with message " + trainingReq.getProcessMessage();

                Response res = new Response(trainingReq,
trainingReq.getProcessRelevance(), return_message + message);
                return res;
        }
}//end of class
```

Figura 54.Código del *plugin* HelloWorld desarrollado en Java (II)

Estos ejemplos sirven para ver cómo realizar el desarrollo de *plugins* en cualquiera de estos tres lenguajes de programación. En caso de querer programar el *plugin* en otro distinto, bastaría con utilizar estas plantillas como recubrimiento y realizar una llamada al ejecutable. Esto por ejemplo es útil para crear un *plugin* remoto de un *plugin* local desarrollado en otro lenguaje de programación distinto de JAVA y que se ha utilizado en el *plugin SpamAssassin*, desarrollado en *bash*. Bastaría con utilizar las siguientes líneas:

Process process = Runtime.getRuntime().exec(command);
int status = process.waitFor();

Donde status sería directamente el valor de relevancia devuelto por el *plugin* y el mensaje se puede leer mediante *process.getInputStream()* y *process.getErrorStream()* en caso de querer saber si ha habido errores. La variable *command* no es otra cosa que una cadena de caracteres con el formato de entrada de los *plugins*. La cadena correspondiente al ejecutable se obtiene mediante *prop.getProperty("pluginExecutable")* y la cadena de argumentos se puede obtener con los métodos *request.getAsString() o request.getAsString(content_file).*La diferencia radica en que el método sin argumentos utiliza el valor del atributo *contentURI* de la petición, por lo que sólo sería válido para ejecuciones locales. Sin embargo, la segunda opción permite pasar el valor a utilizar, por lo que podría utilizarse en el código de las Figura 53 y 54, que es precisamente lo que hace el *plugin SpamAssassin*.

Anexo V: Instalación de un servidor de *plugins* remotos mediante Apache Tomcat y Axis2. Paquete Java para desarrolladores de *plugins*.

En este anexo se cubrirá el proceso de instalación de un servidor de *plugins* remotos, que consiste básicamente en instalar Java, Tomcat y añadirle la funcionalidad Axis. En nuestro caso utilizamos el mismo servidor tanto para el directorio LDAP como para los *plugins*.

Primero de todo, si aún no tenemos Java, se instala mediante el comando:

sudo apt-get install openjdk-6-jdk

En cuanto a Tomcat y Axis, podemos bien instalar Tomcat y añadirle la funcionalidad Axis o bien utilizar directamente el paquete *plugins.tomcat.embedded.jar* que combina una clase java que monta dinámicamente en temporal el servidor con el módulo Axis.

java -jar plugins.tomcat.embedded.jar

Si por el contrario preferimos la instalación de Tomcat, el sistema operativo servidor de Ubuntu ofrece dos posibilidades: si estamos preparando el servidor desde cero podemos instalar Tomcat conjuntamente con el sistema operativo. Si no nos interesa o ya tenemos el sistema operativo, lo instalaremos a posteriori. La diferencia entre ambos son los directorios en que se instalarán los distintos elementos de Tomcat. En cualquiera de los dos casos, tendremos que añadir el módulo Axis manualmente, como se verá en los siguientes apartados.

Instalación de Apache Tomcat durante la instalación del sistema operativo.

La instalación de Tomcat se puede realizar durante la propia instalación del sistema operativo sin más que seleccionarlo en la pantalla en que se nos brinda la oportunidad de instalar algunos paquetes. En nuestro caso se instaló la versión 6 de Tomcat, quedando instalada en los siguientes directorios:

- */usr/share/*:
    - *tomcat6*: Tomcat propiamente dicho.
    - *tomcat6-admin*: dos aplicaciones web (*manager* y *host-manager*) para administrar Tomcat.
    - *tomcat6-docs*: aplicación web para visualizar la documentación de Tomcat de forma local.
    - *tomcat6-examples*: aplicación web para acceder a los ejemplos de *servlet* y *JSP* de Tomcat.
    - *tomcat6-root:* página de bienvenida mostrada al acceder al puerto 8080.
- */var/lib/tomcat6/webapps/*: contiene las aplicaciones web.

- */etc/tomcat6/:* contiene los ficheros de configuración.

Al haber hecho la instalación indicando que se trata de un servidor web, Tomcat se arranca directamente al arrancar el sistema operativo. En cualquier caso es posible arrancarlo y pararlo mediante los scripts *startup.sh* y *shutdown.sh* que se encuentran en la carpeta */usr/share/bin*.

Para comprobar que está correctamente instalado, abrimos un navegador y tecleamos *http://IP_servidor:8080*, ya que Tomcat funciona por defecto sobre el puerto 8080.

Instalación de Apache Tomcat durante la instalación del sistema operativo.

En el caso de que ya tengamos el sistema operativo instalado, lo que hay que hacer es descargar Tomcat de la página oficial *http://tomcat.apache.org/*. Nos vamos a la carpeta donde queramos realizar la instalación, como en el caso anterior */usr/share/* y descargamos allí por ejemplo, la última versión, la 7: elegimos el tar.gz correspondiente al binario *core*, tendremos la opción de mover, renombrar o borrar la instalación fácilmente e incluso podemos tener varias copias a la vez. Si por comodidad renombramos la carpeta a *tomcat6*, a partir de ahora Tomcat estará instalado en */usr/share/tomcat6/*. Dentro del directorio encontramos principalmente:

- *bin/*: contiene los scripts de arranque (*startup.sh*) y parada (*shutdown.sh*)
- *conf/*: contiene los ficheros de configuración
- *webapps/*: aloja las aplicaciones web

Una vez descomprimido, hay que configurar el entorno añadiendo al *PATH* los directorios donde se encuentran instalados Java y Tomcat:

```
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
PATH=$JAVA_HOME/bin:$PATH
export CATALINA_HOME=/usr/share/tomcat6
PATH=$CATALINA_HOME/bin:$PATH
```

Podemos comprobar que está correctamente instalado exactamente igual que para el caso anterior: abriendo un navegador y tecleando *http://IP_servidor:8080*.

Configuración y administración de Tomcat. Instalación del modulo Axis2.

Aunque en nuestro caso mantuvimos el puerto por defecto 8080, es posible cambiarlo por cualquier otro de valor entre 1024 y 65535 en el fichero *server.xml* situado en */etc/tomcat6/* o */usr/share/tomcat6/conf/*, según el tipo de instalación que se haya realizado. Los datos a modificar son el valor del puerto (en negrita):

```
<--
 .....
    Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector port="8080" protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="8443" />
```

Tomcat incluye una aplicación web (paquete *tomcat6-admin*) para administrarlo: desplegar nuevas aplicaciones y ver el estado de las actuales y del servidor, entre otros. Primero hay que configurar un usuario administrador en el fichero *tomcat-users.xml* situado en */etc/tomcat6/* o */usr/share/tomcat6/conf/* según el tipo de instalación que se haya realizado. Hay que añadir la siguiente línea dentro de las etiquetas *<tomcat-user> y </tomcat-user>:*

<user name="admin" password="admin" roles="admin,manager"/>

El perfil manager sirve para la aplicación manager y el perfil admin para la aplicación host-manager. Una vez reiniciado el servicio mediante *sudo service tomcat6 restart*, ya se puede acceder a la aplicación de administración mediante *http://IP_servidor:8080/manager/html* usando el usuario que acabamos de crear.

El primer paso a realizar antes de poder desplegar ningún *plugin* remoto es instalar el módulo Axis. Se puede hacer simplemente copiando el fichero *.war* dentro del directorio o haciéndolo a través de la aplicación de administración. Una vez que ya está instalado, la URL para acceder a una lista de los *plugins* o servicios web disponibles bajo Axis es:

http://IP_servidor:8080/axis2/services/listServices

La URL para acceder a un servicio web concreto es:

http:// IP_servidor:8080/axis2/services/nombre_servicio

Paquete Java para desarrolladores de *plugins*.

Tras las modificaciones sobre el paquete original, el resultado final son tres paquetes:

- *ilip.plugin*: contiene las clases necesarias para crear un *plugin* en Java, tanto remoto como local, entre las que se incluyen la interfaz que debe ser implementada, las que definen los mensajes de petición y respuesta y otras clases con utilidades
- *ilip.plugin.local*: clases necesarias para la ejecución de un *plugin* local.
- *ilip.plugin.remote*: clases necesarias para la ejecución de un *plugin* remoto.

Cada uno de los paquetes se proporciona almacenado en un fichero *.jar* independiente para aprovechar al máximo la separación de funcionalidades:

- plugin_package.jar: paquete *ilip.plugin*.
- local_package.jar: paquete *ilip.plugin.local*.
- remote_package.jar: paquete *ilip.plugin.remote*.

Utilizaremos un ejemplo sencillo para mostrar el proceso de crear un *plugin* local y uno remoto a partir del típico *Hello World*, que en este caso simplemente devuelve relevancia 1 y "*Hello World:* código_hash_del_contenido" como comentario. Partimos de la estructura de ficheros proporcionada (Figura 50) junto con los paquetes y que está compuesta por:

- el *script* básico *compile_plugin_jar_files.sh.*
- *lib/:* contiene el paquete *plugin_package.jar.*
- *config/:* contiene plantillas de los ficheros de configuración para los ficheros *.jar* (*manifest_plugin.mf*) y *.aar* (*manifest_remote_plugin.mf*), para el servicio web (*service.xml*) y para el *plugin* (*plugin.properties*). Será necesario modificar algunos de ellos, como veremos a continuación.
- *src/*: se entrega vacía puesto que dentro se ha de situar una estructura de directorios definida por el nombre del paquete del *plugin* y, dentro de ésta, las clases que contienen su funcionalidad y que hacen uso de las clases del paquete *plugin_package.jar.*

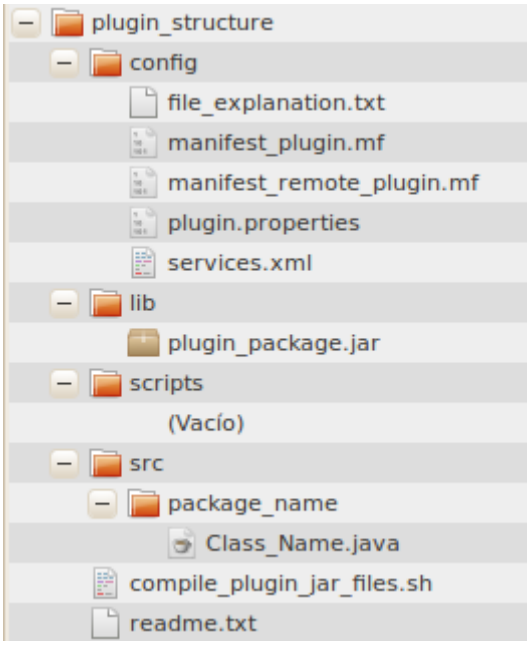


Figura 55. Estructura proporcionada para la creación de plugins.

En el caso de nuestro *plugin* de ejemplo, el paquete se denomina *hw* y sólo existe una clase denominada *HelloWorld*, por lo que bajo el directorio *src/* tenemos la estructura *hw/HelloWorld.java*, donde el fichero contiene el código de las Figuras 48 y 49. En el paquete que creemos para el *plugin* debe haber una clase que implemente la interfaz que se incluye en dicho paquete y que será el punto de entrada por el que reciba los argumentos recibidos en el mensaje SOAP, ya procesados por la librería. Para ello debe implementar los métodos que aparecen en el código de ejemplo.

Los ficheros de configuración almacenan la siguiente información:

- *manifest_plugin.mf*: es el fichero *manifest* del fichero *.jar* a generar. Indica cuál es el paquete y la clase de entrada al *plugin*.
- *manifest_remote_plugin.mf*: es el fichero *manifest* del fichero *.aar* a generar. Indica cuál es el punto de entrada al servicio web.
- *service.xml*: es el fichero de configuración del servicio web. Indica cómo se llama el servicio.
- *plugin.properties*: es el fichero de configuración del *plugin*. Contiene los valores que el *plugin* requiera para su funcionamiento, por lo que su modificación es tarea del administrador del *plugin*.

El *script* plantilla *compile_plugin_jar_files.sh* se proporciona para simplificar la compilación y generación de los ficheros finales, entre ellos los de configuración, aunque el desarrollador es libre de utilizarlo o no:

- El primer argumento consiste en el nombre del paquete y el nombre de la clase separados por un punto: *nombrePaquete.nombreClase*.
- Puede especificarse el tipo de *plugin* a generar mediante las opciones *local* y *remote*. Si no se especifica, se crean ambos.
- Puede especificarse el directorio destino donde almacenar los datos del *plugin* mediante la opción *–d*.
- La compilación genera un fichero *.jar* para un *plugin* local y un fichero *.aar* para un *plugin* remoto.
- Por defecto, el fichero *.jar* que contiene la funcionalidad del *plugin* se denominará *nombrePaquete_nombreClase_plugin.jar* y el *.aar* que incluye el servicio web y que se desplegará bajo el servicio *Axis*, se denominará *nombrePaquete_nombreClase_remote_plugin.aar*.
- Se genera una estructura de directorios para almacenar los datos del *plugin*. Por defecto, si no se ha indicado en los argumentos mediante la opción *-d*, la ruta en que se creará la estructura del *plugin* es /*opt*/ bajo el directorio *nombrePaquete_nombreClase_plugin*.
- Se genera un *plugin* local simplemente copiando los datos del *plugin*, esto es, fichero de configuración y fichero *.jar*, al directorio destino mencionado en el apartado anterior.
- Se genera un *plugin* remoto creando un fichero *.aar* que contiene el fichero *.jar*, el fichero de configuración del servicio web (*services.xml*) y el fichero *remote_manifest.mf* con los valores apropiados, y desplegándolo bajo el servicio *Axis*. Por último, se copia el fichero de configuración del *plugin* al directorio destino.

Tras la ejecución de este *script*, el tipo de *plugin* elegido (en este caso ambos) estará listo para ser configurado en el *Plugin Manager* utilizando las siguientes rutas:

- *Plugin* local: file:///opt/hello_world_plugin/bin/hw_HelloWorld.sh
- *Plugin* remoto:
  http://X.X.X.Y:8080/axis2/services/hw_HelloWorld_plugin

Configuración necesaria en el servidor ILIP para ejecutar *plugins* remotos: instalación del servidor VSFTP de archivos para *plugins* remotos.

Como ya se ha mencionado en varias ocasiones, existe un script de instalación que realiza los siguientes pasos:

- Preparar el sistema instalando todos los paquetes necesarios y configurando las variables necesarias, como por ejemplo, la variable de entorno de Axis para poder realizar llamadas a los *plugins* remotos.
- Preparar el servidor Apache y el servidor de contenidos VSFTP.
- Preparar la librería del módulo distribuidor de contenidos y generar su fichero de configuración con la dirección IP concreta de la máquina.
- Preparar los *plugins* locales.
- Preparar el módulo de traducción VoIP-texto, instalando previamente las herramientas necesarias: Sphinx, SphinxBase, SphinxTrain y PocketSphinx.
- Instalar la herramienta base Xplico, preparando previamente las herramientas necesarias, de momento únicamente Videosnarf.
- Preparar la interfaz gráfica de la plataforma.

A continuación se mencionarán los pasos concretos para instalar y configurar VSFTP. La instalación se realiza mediante el comando:

sudo apt-get install vsftpd

Esto crea el fichero */etc/vsftpd.conf* que deberemos modificar para configurar el servicio. Para no tener que tener un usuario local por cada plugin remoto, empleamos usuarios virtuales mapeados a un único usuario local del sistema, que llamaremos *vsftpd_plugin,* creado mediante el comando:

useradd --home /opt –gid nogroup -shells /bin/false vsftpd_plugin

Para poder acceder a los contenidos los plugins remotos se identificarán mediante un nombre de usuario y una contraseña que *vsftpd* validará. Es necesario almacenar la información usuario-contraseña en algún formato al que el servicio *PAM* pueda acceder mediante el módulo correspondiente. Los que se utilizan más comúnmente son:

- Para una base de datos MySQL se usa el módulo *pam_mysql.so.*
- Para una base de datos de tipo Berkeley se usa el módulo *pam_userdb.so.*
- Para un fichero de tipo *.htpasswd* de Apache, que almacena la información en el formato *usuario:contraseña_cifrada* se usa el módulo *pam_pwdfile.so.*

El problema que se encontró con las bases de datos de tipo Berkeley es que son complicadas de manejar. Se crean a partir de un fichero de texto con la información en claro y para añadir nuevos usuarios a posteriori lo más sencillo es recrear toda la base de datos a partir del fichero añadiendo la nueva información. Esto implica mantener tanto

el fichero en claro como la base de datos, lo cual no es razonable. Por simplicidad se ha escogido usar el tercer método pero si se prefiere utilizar una base de datos MysQL se pueden seguir los pasos proporcionados en http://www.howtoforge.com/virtual-hosting-with-vsftpd-and-mysql-on-ubuntu-12.04. El fichero de tipo *htpasswd* puede crearse mediante el comando *touch* o mediante la opción *–c* la primera vez que se inserte una entrada en el mismo mediante el comando *htpasswd*. Para posteriores inserciones no usar dicha opción. La opción *–d* encripta la contraseña:

sudo htpasswd -c -d /etc/vsftpd_plugin_logins hello_world_plugin

Una vez se ha creado tanto el usuario como el fichero de contraseñas pasamos a configurar los servicios *PAM* y *vsftp*. Para el primero guardamos una copia del fichero original */etc/pam.d/vsftpd* y lo reconfiguramos para utilizar el fichero de contraseñas creado:

```
@include common-session
auth     required     pam_pwdfile.so     pwdfile /etc/vsftpd_plugin_logins
account  required     pam_permit.so
```

En segundo lugar modificamos el fichero de configuración */etc/vsftpd.conf* . Los parámetros que no aparecen en él toman su valor por defecto por lo que en caso de decidir utilizar dicho valor, no será necesario añadirlo. A continuación se explicarán los parámetros que es necesario conocer y los valores que deben tener:

- *anonymous_enable*: activo permite que los usuarios anónimos *anonymous* y *ftp* se conecten. Por defecto está activo pero lo desactivamos por seguridad ya que sólo se deben conectar usuarios autorizados.
- *chroot_local_user*: activo enjaula a los usuarios en su directorio *home*. Por defecto está desactivada. Es necesario activarla cuando se usan usuarios virtuales.
- *dirlist_enable*: si está activo, los usuarios pueden ver los listados de directorios. Por seguridad podemos desactivarla ya que los clientes no lo necesitan.
- *ftpd_banner*: al establecer una conexión se muestra la cadena de caracteres especificada en este parámetro, a no ser que *banner_file* (ruta al fichero que contiene el contenido a mostrar) esté activa, en cuyo éste último caso la sobrescribe. Si ninguna de las dos está presente, se muestra la pancarta estándar de *vsftpd*.
- *guest_enable* y *guest_username*: cuando *guest_enable* está activo todos los usuarios no anónimos se mapean como el usuario especificado en *guest_username*.
- *idle_session_timeout*: especifica la cantidad máxima de tiempo (en segundos) entre comandos desde un cliente remoto. Transcurrido dicho tiempo, se cierra la conexión. Por defecto vale 300 pero podemos darle un valor mucho menor ya que los clientes conectarán y descargarán directamente un único fichero en cada conexión, por lo que nos aseguramos que el sistema cierre la conexión en seguida.
- *listen*: si está activo, VSFTPD se ejecuta en modo independiente y escucha los puertos directamente.

- *local_enable*: activo permite que los usuarios locales del sistema se conecten y también permite configurar usuarios virtuales.
- *local_root*: especifica el directorio al que cambiar una vez que el cliente se ha conectado. Debido a que hemos activado *chroot_local_user* dicha carpeta no debe tener permisos de escritura o el acceso será denegado. Por eso usamos directamente */opt* ya que es el directorio raíz en el que se almacenan los contenidos a los que los *plugins* requieren acceso. Es por ello que éste es el *home* del usuario al que mapean los usuarios virtuales.
- *pam_service_name*: especifica el nombre de servicio PAM que controla el acceso al sistema. Se corresponde con el nombre de un fichero en el directorio */etc/pam.d/* que contiene la configuración del servicio PAM que queremos utilizar. El valor predeterminado es *ftp* por lo que lo cambiamos a *vsftp*d.
- *tcp_wrappers*: si está activo se utilizan TCP *wrappers* para otorgar acceso al servidor.
- *write_enable*: activo permite ejecutar comandos que modifican el sistema de ficheros (crear o eliminar ficheros,…). Por defecto está desactivado.

Como conclusión el fichero */etc/vsftpd.conf* debe contener:

```
anonymous_enable=NO
chroot_local_user=YES
dirlist_enable=NO
ftpd_banner=Welcome to ILIP FTP service
guest_enable=YES
guest_username=vsftpd_plugin
idle_session_timeout=20
listen=YES
local_enable=YES
local_root=/opt
pam_service_name=vsftpd
tcp_wrappers=YES
write_enable=NO
```