TESIS DOCTORAL

PAPI COMO INFRAESTRUCTURA DE SEGURIDAD DISTRIBUIDA APLICADA A ENTORNOS DE FUSIÓN TERMONUCLEAR

RODRIGO CASTRO ROJO
Licenciado en Informática

DEPARTAMENTO DE INFORMÁTICA Y AUTOMÁTICA DE LA ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

U.N.E.D.

2010

DEPARTAMENTO DE INFORMÁTICA Y AUTOMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

U.N.E.D.

PAPI COMO INFRAESTRUCTURA DE SEGURIDAD DISTRIBUIDA APLICADA A ENTORNOS DE FUSIÓN TERMONUCLEAR

RODRIGO CASTRO ROJO
Licenciado en Informática

Director: Dr. D. Jesús Antonio Vega Sánchez

Tutor: Dr. D. Sebastián Dormido Canto

Agradecimientos

Después de estos años de trabajo, es el momento de agradecer la ayuda de toda la gente que me ha apoyado para poder desarrollar la tesis. Durante este tiempo, mi carrera profesional ha tenido dos grandes pilares: mi entorno laboral y mi entorno familiar, y me siento realmente afortunado en ambos casos. A nivel laboral he encontrado grandes profesionales, que también han sido grandes compañeros. De ellos he aprendido, he disfrutado trabajando, y cuando lo he necesitado, he obtenido su confianza para desarrollar ideas y proyectos. De mi familia, siempre he sentido su cariño, apoyo y confianza, sin lo cual no hubiera llegado hasta aquí.

En primer lugar quiero dar las gracias al **Dr. Jesús Antonio Vega Sánchez** por haber confiado en mí, por todo lo que me ha enseñado y aportado como investigador, por su amistad a nivel personal, y por su profesionalidad en la dirección de mi tesis; con buenos consejos, y convirtiendo la exigencia en eficacia y los retos en éxitos.

Quiero dar las gracias al **Dr. Sebastián Dormido Canto** por su apoyo y dedicación durante estos años.

Quiero dar las gracias a mi esposa **Cristina** por estar siempre ahí, por haber confiado siempre en mí, y por motivarme continuamente a mirar hacia delante y a afrontar nuevos retos. Además, quiero agradecerle a nivel profesional, su ayuda en momentos cruciales, y su buen consejo, sin lo cual, no existiría esta tesis. Y quiero que sienta que en gran parte es también suya.

Quiero dar las gracias al **Dr. Diego R. López García** por su apoyo y ayuda desde aquél primer PAPI y a lo largo de todo el tiempo que estuve en RedIRIS.

Quiero agradecer a mis compañeros del grupo de adquisición de datos del TJ-II (CIEMAT), su compañerismo y profesionalidad. En especial a **Augusto Pereira** por su amistad, apoyo y buenos momentos. También agradecer a **Ana Portas** su inestimable ayuda en estos años.

Agradecer al **Dr. Andrea Murari** y al **Dr. Krishan Purahoo** de JET su inestimable ayuda y apoyo técnico en diferentes proyectos, que han sido básicos en el desarrollo de la tesis.

Agradecer al **Dr. Tomas de Miguel** y al **Dr. Víctor Castelo** su confianza en mí y los buenos recuerdos a nivel profesional y personal que conservo de mi paso por RedIRIS. También agradecer la mis **compañeros de RedIRIS** su amistad y el gran recuerdo que conservo de mis años allí.

Quiero agradecer muy especialmente a **mis padres** su infinita confianza y comprensión. Asimismo, quiero agradecerles su esfuerzo por darme una educación y unos valores, fundamentales en vida y en mi carrera profesional, y que espero saber transmitir a mis hijos: **Sergio, Héctor** y **Álvaro**.

Quiero dar las gracias a mi hermana **Marta**, y a mi familia: **Paz, Sonia y Asterio**, por su apoyo y cariño a lo largo de estos años, y por haber estado ahí cuando lo he necesitado.

Quiero agradecer a **Antonio López**, al **Dr. Bernardo Zurro**, y al **Dr. Paco Tabares**, su cordial acogida desde que llegué por primera vez al CIEMAT.

Agradecer al **Dr. Joaquín Sánchez Sanz** su confianza y apoyo durante estos años en CIEMAT.

Agradecer a **Antonio Mollinedo** su ayuda, voluntad y apoyo profesional.

Índice

Lista de símbolos, abreviaturas y siglas

AA - Attribute Authority. Componente de un sistema Shibboleth destinado a facilitar atributos de sus usuarios.

Active Directory - Implementación de Microsoft de un directorio ,que se asocia a un dominio Microsoft, y funciona como repositorio para almacenanr las propiedades de los objetos de dicho dominio.

AEDS - Asynchronous Event Distribution System. Sistema distribuido de sincronización de procesos.

AES - Advanced Encryption Standard. Esquema de cifrado de clave simétrica adoptado como un estándar de cifrado por el gobierno de los Estados Unidos.

Apache - Servidor web desarrollado bajo licencia Open Source.

AQHR - Attribute Query Handle Request. En la arquitectura Shibboleth, consulta enviada por un SHIRE a un HS sobre el SH asociado a un usuario.

AQM - Attribute Query Message. En un sistema Shibboleth, consulta enviada desde el SHAR al AA solicitando ciertos atributos de un usuario.

ATTREQ - Attribute Request. Consulta de solicitud de atributos en una arquitectura PAPI.

ARM - Attribute Release Message. En un sistema Shibboleth, mensaje de respuesta que un AA cuando se le pregunta por los atributos de un usuario.

ARP - Attribute Release Policy. Política de atributos de un usuario. Incluye qué atributos se pueden suministrar sobre ese usuario y con qué fin.

Base64 - Representación de contenido binario mediante un formato tipo texto compuesto por un juego de 64 caracteres.

C - Lenguaje de programación de proóosito general desarrollado en 1972 por Dennis Ritchie en los laboratorios Bell para ser utilizado en el sistema operativo Unix.

CA - Certification Authority. En una infraestructura de clave pública, es una entidad que firma digitalmente certificados y certifica que lo que hay dentro es correcto.

CGI - Common Gateway Interface. Aplicación lanzada por un servidor web, a partir de una consulta HTTP, que es capaz de obtener los parámetros incluidos en dicha consulta, y que genera como salida una respuesta para el cliente en un formato (tipo MIME) reconocible por el servidor web.

CHAP - *Challenge-Handshake Authentication Protocol*. Protocolo de autenticación en Radius.

CIEMAT - *Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas*, Situado en Madrid, España.

Cookie - Marca o testigo digital utilizado en el protocolo HTTP como identificador de sesión entre un cliente y un servidor.

DES - Data Encryption Standard. Algoritmo de cifrado basado en clave simétrica.

Diffie-Hellman - Protocolo que permite el intercambio secreto de claves entre dos partes que no han tenido contacto previo, utilizando un canal inseguro, y de manera anónima (no autenticada).

DNI - *Documento nacional de Identidad*.

DNIe - Versión del documento nacional de identidad en la que se incluye un certificado de usuario que permite la autenticación del mismo, mediante un dispositivo para lectura de tarjetas "Smart Card".

DSA - *Digital Signature Algorithm*. Algoritmo de firma digital estándar para el Gobierno de Estados Unidos.

DSL - Digital Subscriber Line. Familia de tecnologías que aprovechan la conexión telefónica para proveer conexión de banda ancha.

e-Survey - Nombre del sistema implantado dentro del grupo de participación remota de EFDA para la realización de encuestas vía web. Está basado en tecnología "Limesurvey".

EAP - *Extensible Authentication Protocol*. Extensión al protocolo de autenticación Radius con mejoras de seguridad.

EAP-TLS - *EAP-Transport Layer Security*. Estándar que utiliza las extensiones EAP de Radius para crear un canal seguro utilizando tecnología de clave asimétrica y utilizando una certificado digital de cliente para la autenticación del usuario.

eduRoam - Proyecto para la cración de un espacio común WIFI basado en autenticación Radius en entre organiaciones.

EFDA - *European Fusion Development Agreement*. Organización que engloba a laboratorios de fusión en Europa con el objetivo de potenciar la coordinación y colaboración entre ellos.

FORTRAN - Lenguaje de programación comúnmente utilizado en cálculo computacional.

GET - Tipo de consulta HTTP en la que los parámetros de la consulta van explícitamente incluidos en la URL.

Globus - Conjunto de herramientas para la implementación de un sistema Grid.

GMT - Greenwich Mean Time.

GPoA - Group Point of Access. Componente del sistema PAPI que permite agrupar PoAs. También permite agrupar GPoAs.

Grid - Conjunto de recursos computacionales de entornos heterogéneos que se organizan en la consecución de objetivos comunes.

Ethernet - Familia de tecnologías para redes de área local basadas en transmisión de tramas.

HELIAC - Dispositivo de plasmas de fusión, en el cual el centro de las bobinas de campo toroidal no se encuentra en un mismo plano y sigue una línea helicoidal alrededor del conductor central.

HS - Handler Service. Componente de la arquitectura Shibboleth encargado de sumisnistrar, cuando se le consulte, el SH asociado a un usuario.

HTML - HyperText Markup Language. Lenguaje de etiquetas comúnmente utilizado para la construcción de páginas web.

HTTP - HyperText Transfer Protocol. Protocolo base de la aplicación WWW

HTTPS - Hypertext Transfer Protocol Secure. Versión segura del protocolo HTTP, en base a certificados de servidor (y opcionalmente también de cliente).

IAA - Infraestructura de Autenticación y Autorización.

IDL - Interactive Data Language. Lenguaje de programación orientado al análisis de datos.

IETF - Internet Engineering Task Force. Organismo internacional que promueve el desarrollo de estándares para Interent.

IP - Internet Protocol. Protocolo telemático para la comunicación de datos a través de una red informática de paquetes conmutados.

IST - Instituto Superior Técnico. Organización el la que se engloba el laboratorio de fusión en Portugal.

ITER - Internacional Thermonuclear Experimental Reactor. Máquina de fusión experimental en fase de construcción y que constituye el paso previo al primer reactor de fusión nuclear de demostración.

JAVA - Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems. Una de sus principales características es que el código objeto que genera, funciona en diferentes plataformas.

JavaScript - Lenguaje de programación incluido en navegadores web de tip interpretado.

JET - *Joint European Torus*, tokamak situado en Abingdon, Inglaterra.

JETTY - Plataforma para implementación de servicios HTTP en JAVA.

JNLP - *Java Network Launching Protocol*. Protocolo que permite definir las condiciones y parámetros de descarga y ejecución de una aplicación JAVA y las librerías necesarias.

JWS - *Java Web Start*. Tecnología que permite descargar, actualizar y ejecutar aplicaciones JAVA que se encuentran en repositorios remotos utilizando el navegador web.

KDC - *Key Distribution Center.* Componente del sistema "Kerberos" utilizado para la distribución de un "token" de sesión.

Kerberos - Sistema de autenticación y autorización desarrollado en Internet2.

LabView - Lenguaje de programación de National Instruments comúnmente utilizado para sistemas de control y sistemas embebidos.

LDAP - *Lightweight Directory Access Protocol*. Protocolo de consulta para servidios de directorio en Internet. Se utiliza también para hacer referencia a un servicio de directorio que soporte el protocolo LDAP.

LNFCM - *Laboratorio Nacional de Fusión por Confinamiento Magnético*, situado en las instalaciones del CIEMAT en Madrid, España.

Limesurvey - Sistema con licencia de código abierto para la implementación de encuestas vía web.

Location - Zona de un servidor web

Loggin - Termino referente al inicio de sesión de acceso a us recurso.

Logout - Termino referente al cierre de una sesión de acceso a un recurso.

MathLab - Entorno de calculo computacional desarrollado por la empresa MathWorks.

MDSplus - Sistema desarrollado en el MIT para el acceso a datos experimentales en el entorno de fusión.

MIME - *Multipurpose Internet Mail Extensions.* Convención para unificar los identificadores asociados a formatos de contenidos en Internet.

MIT - *Masachuset Institute of Technology*.

MS-CHAP - *Microsoft CHAP*. Incorpora mejoras al protocolo CHAP original.

MySQL - Gestor de base de datos relacionales.

NAS - *Network Access Service*. Componente de la arquitectura Radius que habilita la conexión de un usuario a la red, previa autenticación.

NASA - National Aeronautics and Space Administration.

NEES - Network for Earthquake Engineering Simulation. Proyecto de la " National Science Foundation" para conocer mejor los movimientos sísmicos y susu efectos.

OASIS - Organization for the Advancement of Structured Information Standards. Organización orientada a estandarizar aspectos relacionados con el área del comercio electrónico y los servicios web.

OpenID - Sistema distribuido de autenticación basado en que el usaurio suministra al propietario de un recurso, si a sí lo requiere, una URL donde le pueden identificar.

PAP - *Password Authentication Protocol*. Protocolo de autenticación en Radius.

PAPI - *Punto de Acceso a Proveedores de Información*. Infraestructura distribuida de autenticación y autorización que garantiza un acceso seguro a los servicios y contenidos soportados.

PEAP - *Protected Extensible Authentication Protocol*. Protocolo basado en la utilización de EAP y certificado en el servidor de autenticación para crea un canal seguro que garantice el proceso de autenticación.

PHP - Lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

PIN - *Personal Identification Number*. Número secreto asociado que sólo conoce el usuario y permite su identificación.

PKI - *Public Key Infrastructure.* Sistema jerárquico de gestión de certificados basados en tecnologías de cifrado de clave asimétrica.

PoA - *Punto de Acceso*. Componente de la arquitectura PAPI encargado de controlar el acceso a un recurso protegido.

POST - Tipo de consulta HTTP en la que el valor de los parámetros se pasan a través de la conexión y no expuestos explícitamente en la URL de la consulta.

Proxy - Concepto utilizado en redes de comunicación telemáticas para referirse a un elemento que realiza las funciones de intermediario en una transacción entre un cliente y un servicio.

Radius - *Remote Authentication Dial In Service*. Protocolo de autenticación sobre IP. El termino se utiliza para referirse a servidores de autenticación basados en dicho protocolo.

RBNB - *Ring Buffering Network Bus*. Tecnología que permite la distribución de flujos de datos de forma eficiente.

RC4 - Algoritmo de cifrado basado en clave simétrica habitualmente utilizado como protocolo base de otros como SSL.

RedIRIS - Red española para Interconexión de los Recursos Informáticos de las universidades y centros de investigación.

REST - Representational State Transfer. Paradigma para la implementación de servicios web basado en consultas HTTP tipo GET.

RM - *Resource Manager*. Elemento de un sistema Shibboleth encargado de controlar el acceso a un recurso.

RPC - *Remote Procedure Call.* Protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota

RSA - *Rivest, Shamir y Adleman*. Apellidos de los creadores de uno de los algoritmos de clave asimétrica más utilizados.

Single-Sign-On - Termino referido a la capacidad de un sistema de seguridad de, una vez autenticado, un usuario poder acceder a diferentes recursos de forma segura a lo largo de una sesión, sin necesidad de volverse autenticar.

SA - *Servidor de Autenticación*. Componente de la arquitectura PAPI, encargado de autenticar usuarios.

SAML - *Security Assertion Markup Language*. Protocolo diseñado para sistemas de autenticación, autorización y registro.

SDAS - Shared Data Access System

SH - *Shibboleth Handler*. Identificador manejado en un sistema Shiboleth para mantener la sesión de un usuario.

SHA - Secure Hash Algorithm. Familia de algoritmos "hash" con cifrado diseñados por la Agencia Nacional de Seguridad de Estados Unidos.

SHAR - Shibboleth Attribute Requester. Componente del sistema Shibboleth destinado a realizar consultas sobre atributos del usuario.

SHIRE - *Shibboleth Handler Requester*. Elemento de la arquitectura Shibboleth encargado de consultar a un HS sobre el Shibboleth handler asociado a un usuario.

"Smart Card" - Tarjeta portátil (del tamaño de una tarjeta de crédito) que incluye un microprocesador y en la que se pueden incluir funcionalidades criptográficas.

Sockets - Elemento virtual utilizado a nivel de programación para la comunicación de aplicaciones y procesos a través de IP.

SPNEGO - Simple and Protected GSSAPI Negotiation Mechanism. Solución que permite negociar el mecanismo de autenticación dentro del protocolo HTTP.

SQL - *Structured Query Language*. Lenguaje de consulta para bases de datos relacionales. Se utiliza también para referirse a un servicio en Internet de base de datos realcional que soporte este lenguaje de consulta.

SSL - Secure Sockets Layer. Protocolo de conexión segura basado en intercambio de clave simétrica de sesión utilizando criptografía de clave pública.

STAgent - Componente de la arquitectura PAPI que permite a las aplicaciones solicitar un "token" de servicio.

STELLARATOR - Dispositivo de confinamiento magnético. El nombre deriva de "stella generator", o lo que es lo mismo, generador de estrellas en clara alusión a los futuros reactores de fusión como pequeñas estrellas en la tierra.

TCP - Transmission Control Protocol. Protocolo basado en IP para comunicación orientada a conexión

TJ-II - *Torus JEN-II*, Nombre con el que se denominó al dispositivo de plasmas de fusión tipo *stellarator* heliac flexible instalado en el LNFCM.

Ticket - El mismo concepto que el "token" pero en terminología "Kerberos".

Token - Termino comúnmente utilizado en entornos de seguridad para referirse a un testigo en formato digital.

Tomcat - Servidor de aplicaciones JAVA que facilita a las aplicaciones, entre otras cosas, el manejo del protocolo HTTP para las consultas y respuestas.

UDP - *User Datagram Protocol*. Protocolo basado en IP para comunicación no orientada a conexión.

UNIX - Sistema operativo portable, multitarea y multiusuario

URL - *Uniform Resource Locator*. Expresión en formato texto que permite referenciar recursos y contenidos localizados en Internet.

USB - *Universal Serial Bus*. Puerto que sirve para conectar periféricos a un computador.

VPN - *Virtual Private Network*. Tecnología que permite crear canales seguros de conexión utilizando protocolos IP.

VSA - *Vendor Specific Attributes*. Atributos que los diferentes fabricantes pueden incluir en el protocolo.

WAYF - *Where Are You Form*. Componente de un sistema de autenticación y autorización distribuido encargado de consultar, o descubrir mediante alguna técnica, la organización origen a la que pertenece el usaurio.

VxWorks - Sistema operativo UNIS de tiempo real vendido y fabricado por Wind River Systems.

WYSIWYG - What You See Is What You Get. Termino utilizado para definir entornos visuales en los que el contenido resultante es tal cual lo muestra la aplicación.

Web - Termino equivalente a WWW.

WIFI - *Wireless Fidelity*. Es un sistema de envío de datos sobre redes informáticas que utiliza ondas de radio en lugar de cables.

WIKI - Recurso web con herramientas visuales de edición que permiten a usuarios crear de forma sencilla sus propias páginas web.

Windows - Familia de sistemas operativos desarrolados pro Microsoft y orientados a ordenadores personales.

WWW - *World Wide Web*. Red de contenidos localizados y accesible a través de Internet que se enlazan mediante expresiones URL

X.509 - Formato de documento digital utilizado comúnmente para certificados digitales basados en algoritmos de clave asimétrica.

XML - *Extensible Markup Language*. Lenguaje de formato para ficheros texto que permite estructurar los datos almacenados haciéndolos útiles para la computación y entendibles para los humanos.

XWIKI - Implementación Open SOurce de un servicio Wiki.

Índice de figuras

Capítulo 1

Introducción

Con el avance en las comunicaciones y la integración de los sistemas informáticos en Internet, los usuarios han pasado de trabajar en sistemas cerrados, (donde estaciones de trabajo, servicios y recursos se encontraban aislados a nivel de comunicación), a sistemas abiertos, donde los clientes y proveedores de servicios se encuentran conectados a Internet en redes diferentes. Desde el punto de vista de los recursos y servicios, su distribución en diferentes organizaciones, no debería suponer una dificultad a la hora de acceder a los mismos. En la actualidad, los entornos y aplicaciones de usuario tienden a enmascarar la localización y acceso a los diferentes servicios. En consecuencia, el usuario percibe y trata los recursos como locales, sin importarle ni la localización de los servidores implicados, ni los métodos de acceso a cada uno de ellos, y por supuesto, evitando el tener que conocer diferentes contraseñas para acceder a cada uno de ellos. Del mismo modo, pero desde el punto de vista del cliente, la capacidad de acceder a los servicios y recursos debe ser independiente de la localización del usuario y del método de conexión a Internet que utilice.

Este tipo de entornos abiertos, ha disparado la necesidad de integrar y aplicar soluciones de seguridad a sus sistemas de información. Dichas soluciones deben ser capaces de trabajar en sistemas que engloban a varias organizaciones, y por tanto, deben ser suficientemente flexibles como para ser integrados en sus sistemas de información. Además, deben ser soluciones escalables, independientemente del número de organizaciones, usuarios y servicios que engloben. Por último, deben ser compatibles con la movilidad de los usuarios y flexibles con la localización de los servicios. Por todo ello, soluciones de seguridad que resultan eficaces en sistemas cerrados (normalmente soluciones centralizadas o basadas en la dirección IP origen de las

conexiones), resultan poco flexibles, además de difíciles de mantener y gestionar en este tipo de entornos.

PAPI (Punto de Acceso a Proveedores de Información) es una solución de seguridad concebida, desde un principio, para dar respuesta a este tipo de sistemas abiertos. Su principal característica es su arquitectura distribuida que le permite posicionar sus elementos y procesos en diferentes puntos de la red independientemente de su localización. Esta característica le da una gran capacidad de integración en entornos multi-organización, ya que permite distribuir sus funciones, y simplifica la gestión global de la infraestructura de seguridad y potencia su escalado.

El TJ-II es una máquina tipo ESTELLARATOR perteneciente al Laboratorio de Fusión por Confinamiento Magnético del CIEMAT (Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas). Tanto el TJ-II, como otros dispositivos experimentales, requieren de soluciones que garanticen la seguridad en los accesos de los usuarios a los diferentes servicios y aplicaciones. Asimismo, estas soluciones deben poderse adaptar a un esquema colaborativo en el que los usuarios pertenecen a diferentes organizaciones, y en el que sus localización a nivel de red no es fija, debido a la naturaleza de sus actividades. En este trabajo se describe, tanto el proceso de implantación de PAPI como solución de seguridad en este tipo de entornos, como todo el conjunto de desarrollos e innovaciones tecnológicas resultantes, en el ámbito de las infraestructuras de seguridad multi-organización.

La estructura de este documento refleja cronológicamente el trabajo desarrollado. Dicho trabajo no parte de cero, ya que previamente, el autor fue uno de los creadores del sistema PAPI, participando desde su concepción en la red de investigación española RedIRIS, en el año 2001 [1]. En el capítulo 2 se realiza una descripción detallada del sistema PAPI resultante de este primer desarrollo, y muestra su arquitectura ,tal y como era, antes de comenzar su implantación en el TJ-II.

En el capítulo 3 se detalla el proceso de implantación de PAPI como infraestructura de seguridad del entorno de participación remota del TJ-II. Al comienzo del capítulo se describe el problema a resolver. A continuación, se realiza un análisis comparativo de varias soluciones se seguridad teniendo en cuenta los requisitos del problema planteado. Finalmente se detallan aspectos técnicos relacionados con el proceso de implantación, enfatizando el conjunto de mejoras desarrolladas para PAPI y que suponen

innovaciones tecnológicas en el ámbito de las infraestructuras de autenticación y autorización distribuidas.

Para dar respuesta a la cada vez más creciente demanda de colaboración entre laboratorios de fusión, y a la necesidad de compartir recursos y servicios de forma segura, desde CIEMAT arranca una iniciativa, pionera en el entorno de fusión, para la creación de una estructura organizativa federada entre laboratorios pertenecientes a EFDA (European Fusion Development Agreement), en base a una infraestructura de seguridad multi-organización. En el capítulo 4 se describe dicha iniciativa, presentando al comienzo del mismo, un análisis comparativo entre un conjunto de soluciones posibles. A continuación, se detallan aspectos técnicos relevantes del proceso de desarrollo, y se vuelve a enfatizar en los resultados que suponen innovaciones tecnológicas en el ámbito de la implementación de federaciones de seguridad.

Finalmente, el capítulo 5 expone un conjunto de conclusiones y líneas futuras de trabajo que permitan explotar los avances obtenidos.

1.1. Sistemas de seguridad de control de acceso

1.1.1. Funciones básicas de los sistemas de control de acceso

Los sistemas de control de acceso se utilizan, principalmente, con el objetivo de garantizar dos aspectos del acceso a recursos. En primer lugar, que el usuario que accede a un recurso es identificable, y además que la identidad se corresponde con el usuario que hace uso de ella. Esto se consigue mediante un proceso de autenticación. Existen muy diversos mecanismos para autenticar a un usuario:

- Sistemas basados en algo conocido. Ejemplo, una contraseña que sólo debe conocer dicho usuario.
- Sistemas basados en algo que posee el usuario. Por ejemplo: una tarjeta de identidad, una tarjeta inteligente (“smart card”), un dispositivo criptográfico USB (Universal Serial Bus).
- Sistemas basados en una característica física del usuario o un acto involuntario del mismo: Ejemplo, verificación de voz, de escritura, de huellas, de patrones oculares.

Además de lo que es la pura comprobación de que un usuario es quien dice ser, existe cada vez más la necesidad de incluir características del usuario dentro de la autenticación (llamados atributos), que se organizan en una estructura denominada aserción. De esta forma, además de asegurar que el usuario se corresponde con una cierta identidad, también se avala el contenido de su aserción.

El segundo aspecto a garantizar en un sistema de control de acceso es que sólo acceden a la información aquellos usuarios que están autorizados para ello. Esto se consigue mediante la definición de políticas de acceso a recursos. Estas políticas de acceso se definen mediante un conjunto de reglas de control de acceso que valoran diferentes criterios relacionados con la consulta al recurso, entre las que se encuentran: datos incluidos en la aserción del usuario (cargo, proyectos de investigación, etc.), información relativa al recurso que se va a acceder y a los datos que le son enviados en la consulta (acción a realizar, objetivo de la acción, etc.), información de entorno de la consulta (fecha y hora, dirección IP origen de la consulta, etc.). Los sistemas de control de acceso permiten implementar las reglas de acceso a un recurso mediante preposiciones lógicas en las que se utilizan operadores (matemáticos, de cadenas de caracteres, fecha y hora, etc.) y parámetros, que representan las características antes comentadas. Además, se utilizan operadores lógicos como "AND", "OR" y "NOT".

Una última característica de estos sistemas es su capacidad de garantizar la seguridad en todo su ámbito de aplicación. Esto implica, por un lado, la seguridad a nivel de los componentes que lo conforman (que queda fuera del ámbito de este trabajo), y por otro lado, la seguridad de las comunicaciones entre diversos elementos. Este último aspecto se garantiza en base a una serie de propiedades de la comunicación:

- Autenticidad: debe poderse asegurar que el mensaje es enviado y recibido por los elementos previstos.
- Integridad: que el contenido de la comunicación no ha sido alterado.
- Cifrado: que el contenido es sólo visible para el emisor y el receptor de los datos.

1.1.2. Secuencias de autorización (agent, pull, push)

Existen diferentes métodos en los que el usuario, la infraestructura de autenticación y autorización y los recursos interactúan durante una transacción. Concretamente, tal y como muestra la *Fig. 1-1*, hay tres tipos de secuencias de interacción que se pueden dar:

Secuencia de tipo agente. En este tipo de secuencia la infraestructura de autorización actúa como mero intermediario. En un primer paso el usuario solicita un servicio a la infraestructura, y ésta, tras autorizar la solicitud, comunica al propietario del recurso la activación del servicio para el usuario en concreto. A continuación se le comunica al usuario la disponibilidad del servicio para su uso.

Secuencia de tipo petición (PULL). En este caso, el usuario conecta directamente con el servicio y solicita su uso. El servicio consulta con la infraestructura de autorización sobre la petición realizada por el usuario. Si todo es correcto, la infraestructura de autorización responde afirmativamente al servicio, que a continuación aprueba la petición del usuario.

Secuencia de tipo envío (PUSH). En este tipo, el usuario conecta en un primer paso con la infraestructura de autenticación. Tras su correcta identificación, el servicio de autenticación le envía una prueba de autenticación, normalmente un testigo ("token") digital firmado, que debe entregar junto con las peticiones de servicio. El propietario del servicio, cuando recibe una petición por parte del usuario, comprueba el testigo adjunto a la petición, y si todo es correcto, da paso al servicio.

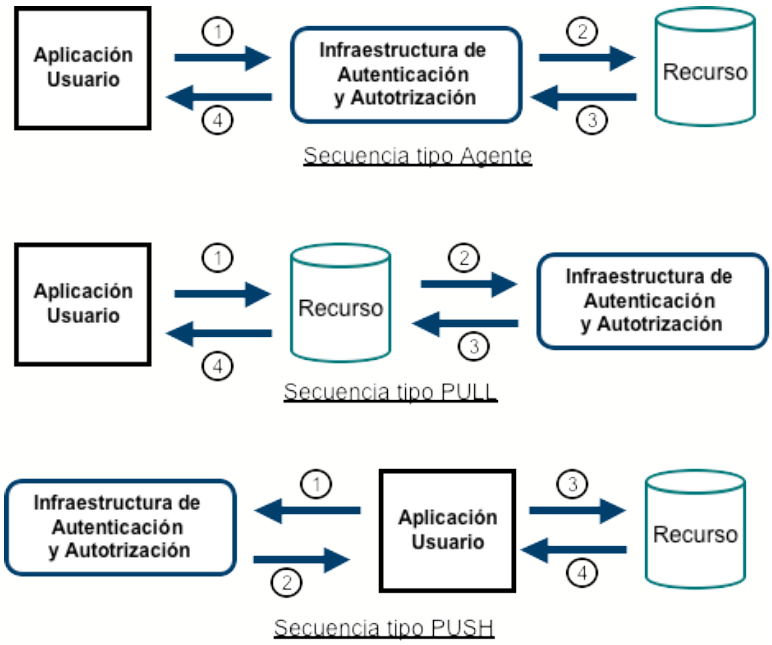

Fig. 1-1 *Diferentes esquemas de autorización en función del tipo de secuencia en el proceso de autorización*

1.1.3. Arquitectura centralizada y distribuida

Dependiendo de la localización de los diferentes procesos de un sistema, y por tanto, de los componentes que los llevan a cabo, se habla de sistemas centralizados o distribuidos, pudiendo haber diferentes grados de una y otra característica. El primer caso corresponde a sistemas en que la mayoría de las funciones y componentes se encuentran localizados en un mismo elemento de proceso. En el caso de sistemas de control de acceso, se consideran soluciones centralizadas, tal y como muestra la *Fig. 1-2* aquellas que unen en un mismo componente las funciones del sistema, y más específicamente, las funciones de autenticación y autorización y además resulta imposible, por concepción de la solución, su separación. De tal forma, que es el mismo elemento el que realiza la autenticación de los usuarios y la autorización en los accesos a los recursos. Este esquema suele ajustarse a sistemas cerrados en los que no actúan elementos externos al sistema y las comunicaciones se realizan en entornos controlados. Este tipo de sistemas suelen presentar problemas de escalado, cuya solución suele pasar por incrementar la capacidad de sus componentes.

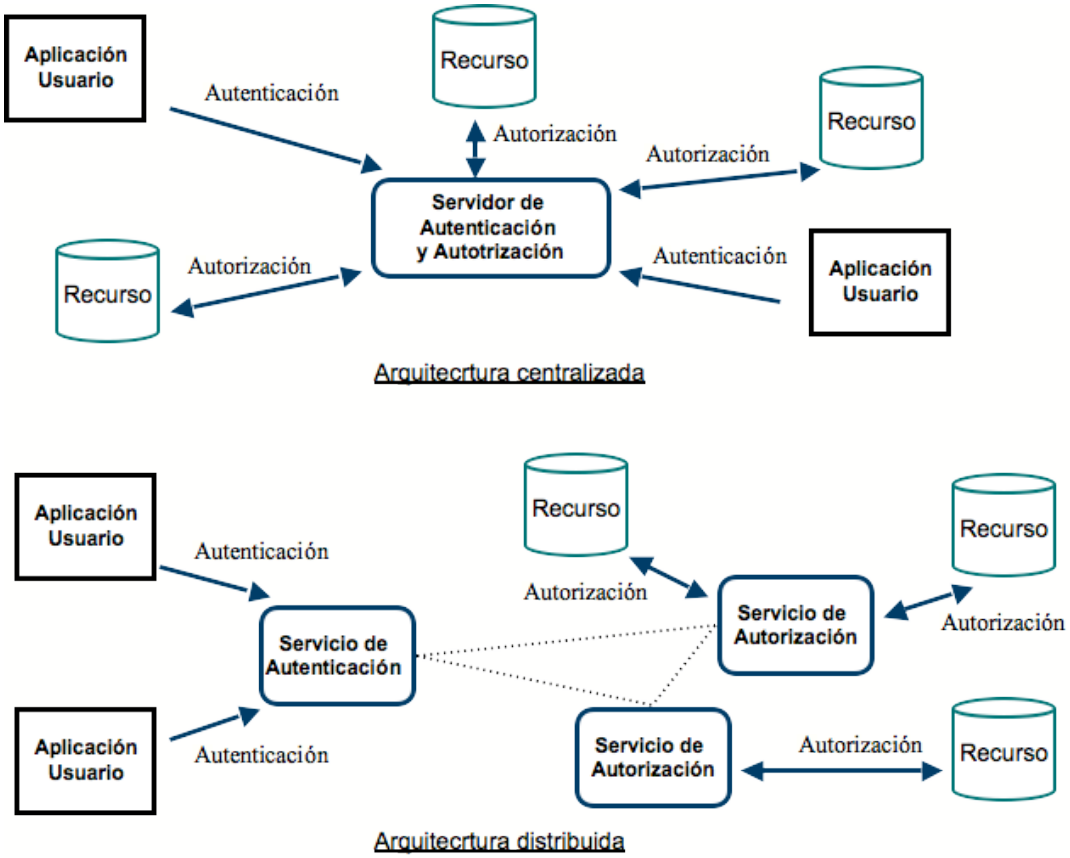

Fig. 1-2 *Arquitectura centralizada y distribuida en infraestructuras de autenticación y autorización*

Por otro lado hay sistemas de control de acceso distribuidos, en los que las diferentes funciones del sistema, y más concretamente la autenticación y la autorización se pueden realizar en elementos diferentes, que incluso se encuentran distantes a nivel de comunicaciones. Este tipo de sistemas incluyen una mayor complejidad a nivel de comunicaciones, pero por contra, son más flexibles para su integración en sistemas ya existentes, facilitan la gestión en entornos abiertos (por ejemplo la gestión de usuarios y recursos en entornos multi-organización), y escalan mejor que los centralizados (siempre que no tengan gran cantidad de tráfico de comunicaciones), ya que sus funciones pueden ser distribuidas.

Capítulo 2

PAPI

2.1. Introducción

En los últimos años el acceso a recursos restringidos dentro de Internet (bibliotecas, publicaciones, bases de datos específicas, etc.) se ha hecho cada vez más común. En la inmensa mayoría de los casos, se ha utilizado como método de control de acceso filtros sobre la dirección IP origen de la consulta, demostrándose claramente ineficiente en muchos casos.

Este es un problema que afecta a la mayoría de organizaciones de investigación a la hora de acceder a recursos restringidos en Internet. PAPI [1,23] comenzó siendo una propuesta de solución técnica en una reunión a la que asistieron representantes, tanto de consorcios de bibliotecas localizados dentro de la red I+D, como de empresas proveedoras de información por Internet. La solución final tendría que cumplir una serie de requisitos como: ser lo más transparente posible, compatible con los navegadores y servidores web más comunes, que no necesitara de la instalación de software o hardware adicional, y que permitiera el acceso al usuario independientemente de su localización.

El modelo de PAPI ha evolucionado en cuanto a su implementación, pasando de ser una solución basada en certificados temporales a ser un sistema basado en "cookies" cifradas. En la actualidad, su uso cubre varios frentes, desde servir como sistema para el acceso a proveedores de información de numerosos centros docentes y de investigación, hasta ser utilizado como infraestructura de autenticación y autorización para el

desarrollo de sistemas distribuidos entre centros de investigación, permitiéndoles compartir datos y aplicaciones.

2.1.1. Requisitos iniciales de PAPI

El desarrollo de PAPI está basado en un conjunto de requisitos que definen las principales características de este sistema a nivel funcional:

- Acceso independiente de la IP origen. El sistema de control de acceso debe ser independiente de la IP origen de la conexión.
- Una vez realizada la autenticación, el usuario debe tener acceso a todos los recursos durante un período limitado de tiempo. El sistema debe ser capaz de autenticar al usuario, y una vez autenticado, el usuario tendrá acceso a todos los recursos sin requerírsele nuevas autenticaciones. Es decir, se requiere un sistema de autenticación única ("Single-Sign-On"). Además las sesiones deben tener un límite temporal que puede ser fijado.
- Movilidad del usuario garantizada. Los usuarios podrán acceder a los recursos a los que están autorizados independientemente del sistema de conexión empleado o su localización en la red. Los procesos del sistema serán iguales para usuarios locales o remotos.
- Transparente para el usuario. El sistema debe ser lo más transparente para el usuario que sea posible, requiriéndole una interacción mínima.
- Compatible con otros sistemas de control de acceso. El sistema debe poder ser utilizado en conjunción con otros sistemas de control de acceso ya existentes.
- Compatible con los navegadores más usados. La solución debe ser compatible con los navegadores web más habituales: MS Explorer, Mozilla, Opera y Safari.
- Garantizar la privacidad del usuario. La solución debe implementar mecanismos que garanticen el anonimato en el acceso a ciertos servicios.
- Facilitar las estadísticas en los proveedores. El sistema debe garantizar que los accesos puedan ser discriminados por usuario, para poder obtener estadísticas de acceso fiables.

2.2. Arquitectura básica del sistema PAPI

El sistema, como puede verse en la Fig. 2-1, consta de dos partes: el servidor de autenticación (SA) y el punto de acceso (PoA). El primero de ellos se encarga de la autenticación del usuario, mientras que el segundo se encarga de la fase de autorización en el acceso al recurso. Ambos elementos son independientes, no existiendo ninguna restricción respecto a número o localización de cualquiera de ellos. Gracias a esta propiedad el sistema final resulta mucho más flexible y configurable.

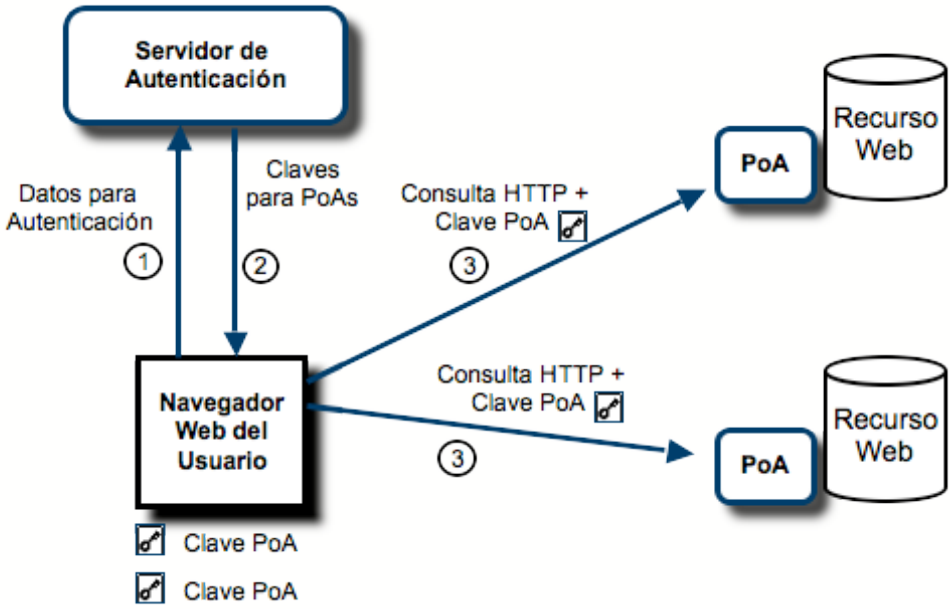

Fig. 2-1 *Esquema básico de funcionamiento de PAPI*

El SA, encargado de la autenticación del usuario, una vez que ha comprobado que el usuario es quien dice ser, también comprueba a qué recursos tiene acceso. Para cada uno de estos recursos, el SA le entrega al usuario (más concretamente a su aplicación web) una clave temporal (el periodo de validez es configurable) cifrada que le permitirá más adelante acceder al recurso en cuestión.

Esta clave se puede asemejar a una llave con un tiempo de validez. Al final de la autenticación, el usuario tendrá en su poder una llave para cada uno de los recursos para los que él está autorizado a entrar. En PAPI estas "llaves" se implementan mediante “cookies” [4]temporales cifradas.

El PoA se encarga de realizar el control de acceso al recurso restringido. Cuando el usuario realiza una consulta a un recurso protegido por un PoA, éste comprueba si el usuario posee una clave válida y si ésta ha sido emitida por un SA válido. Si es así, le deja pasar y cada cierto tiempo (tiempo de rotación) le entrega al usuario una nueva clave para seguir entrando. El tiempo de rotación se estable a nivel de configuración de cada PoA y equivale al tiempo de validez de la cookie "Lcook", que se describe en detalle en el punto 2.2.2. Este procedimiento de refresco sirve para evitar que dos usuarios entren con la misma "cookie", es decir, evita la copia de "cookies" entre equipos. En realidad los PoAs son como cerraduras, y sólo aquellos usuarios que poseen la llave ("cookie") correcta pueden pasar.

Una vez que la clave temporal ha vencido, el usuario tiene que volver a autenticarse para poder conseguir nuevas claves. La utilización de claves temporales permite el no tener que gestionar listas de revocación, como ocurre por ejemplo en el caso de certificados digitales, que son validos durante un largo periodo de tiempo, y la única forma de invalidarlos es a través de listas de revocación, que es un proceso engorroso y no resuelto del todo.

2.2.1. Servidor de autenticación

El servidor de autenticación, tal y como muestra la *Fig. 2-2*, consta de 3 módulos.

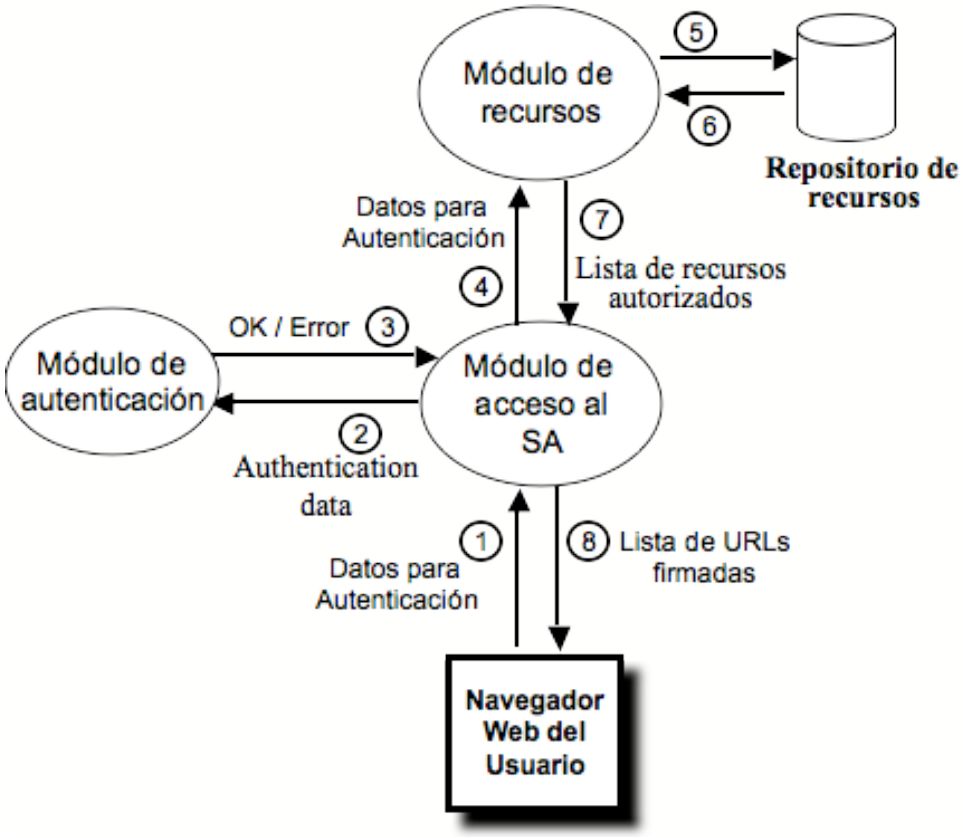

Fig. 2-2 *Arquitectura del servidor de autenticación de PAPI*

Módulo de acceso al SA: Sirve de elemento de control para el procesamiento de las peticiones de autenticación desde la aplicación cliente del usuario, coordinando el acceso al resto de subsistemas del servidor. Sus funciones principales son:

- Recoger la petición de autenticación junto con la información necesaria para ésta.
- Controlar el proceso de autenticación haciéndolo pasar por las diferentes fases y construir la aserción del usuario, utilizando para ello los datos obtenidos en los módulos de autenticación y recursos.
- Construir la respuesta mediante las plantillas HTML personalizadas para cada organización y enviarla de vuelta al usuario, controlando posibles redirecciones en función de los parámetros de la consulta. Como parte de la respuesta se incluye una lista de URLs (una por cada recurso del que el usuario necesite "cookies" PAPI de sesión). En estas URLs se encuentra codificado el mensaje que el servidor de autenticación envía a cada PoA solicitando "cookies" PAPI de sesión para el usuario. Esta firma se realiza

utilizando el algoritmo de clave pública RSA (Rivest, Shamir y Adleman), y como clave de firma, la clave privada del servidor SA. Los PoAs que confíen en este SA, dispondrán de la parte pública de la clave, para así poder verificar la firma de las URLs que reciba desde este SA.

Módulo de autenticación: Se encarga de realizar la verificación de los datos de autenticación proporcionados por el usuario, en función del método de autenticación configurado. Están disponibles módulos de autenticación para: LDAP (Lightweight Directory Access Protocol) [5], SQL (Structured Query Language) [6], "Radius" [7], certificados X.509 [8], ficheros texto, y base de datos Berkeley. Incluso se puede asociar un módulo personal desarrollado a medida.

Módulo de recursos: Se encarga de comprobar a qué recursos tiene acceso el usuario, así como atributos e información necesaria para la construcción de la aserción del usuario, la cual será enviada en la respuesta para hacerla llegar a los PoAs correspondientes. Al igual que el módulo de autenticación, están disponibles módulos para repositorios basados en: LDAP, SQL, ficheros texto, y base de datos Berkeley [9]. También se puede asociar un módulo personal desarrollado a medida.

2.2.2. El punto de acceso

El punto de acceso, como ya se ha comentado, es el elemento encargado de controlar el acceso a los recursos, bien locales, bien remotos que implementa PAPI. En la Fig. 2-3 se muestra un diagrama de la arquitectura del punto de acceso. En él se pueden observar los siguientes componentes:

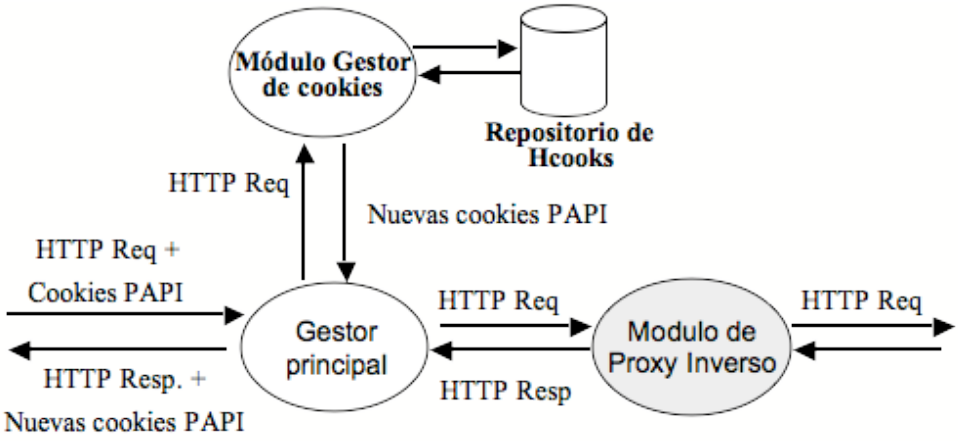

Fig. 2-3 *Arquitectura del Punto de Acceso*

Gestor principal: Es el componente principal del punto de acceso. En su diagrama de actividad, representado en la *Fig. 2-4*, se peden observar sus principales tareas:

- Analizar el tipo de consulta: Debe discriminar las peticiones relacionadas con la gestión de las "cookies" PAPI (comienzo de sesión y generación de nuevas "cookies" PAPI, finalización de sesión y borrado de las "cookies" PAPI, y consulta del estado de las "cookies" PAPI para ese PoA), del resto de consultas propias del servicio al que controla.
- Tanto para consultas relacionadas con la gestión de "cookies" PAPI, como para aquellas situaciones en las que se requiera algún tipo de acción sobre las "cookies", el gestor principal utiliza el módulo gestor de "cookies".
- En función del resultado de las comprobaciones llevadas a cabo por el gestor principal, éste, bien deja pasar la consulta para poder ser resuelta en el servicio asociado, bien la envía al módulo de "proxy" inverso (si el servicio se encuentra en una localización remota), bien deniega el acceso al servicio.
- A la respuesta HTTP (HyperText Transfer Protocol) [10] de la consulta del usuario, el gestor principal añade, en caso de así requerirlo, las nuevas "cookies" generadas. Con ello se garantiza que las siguientes consultas contendrán las nuevas "cookies".

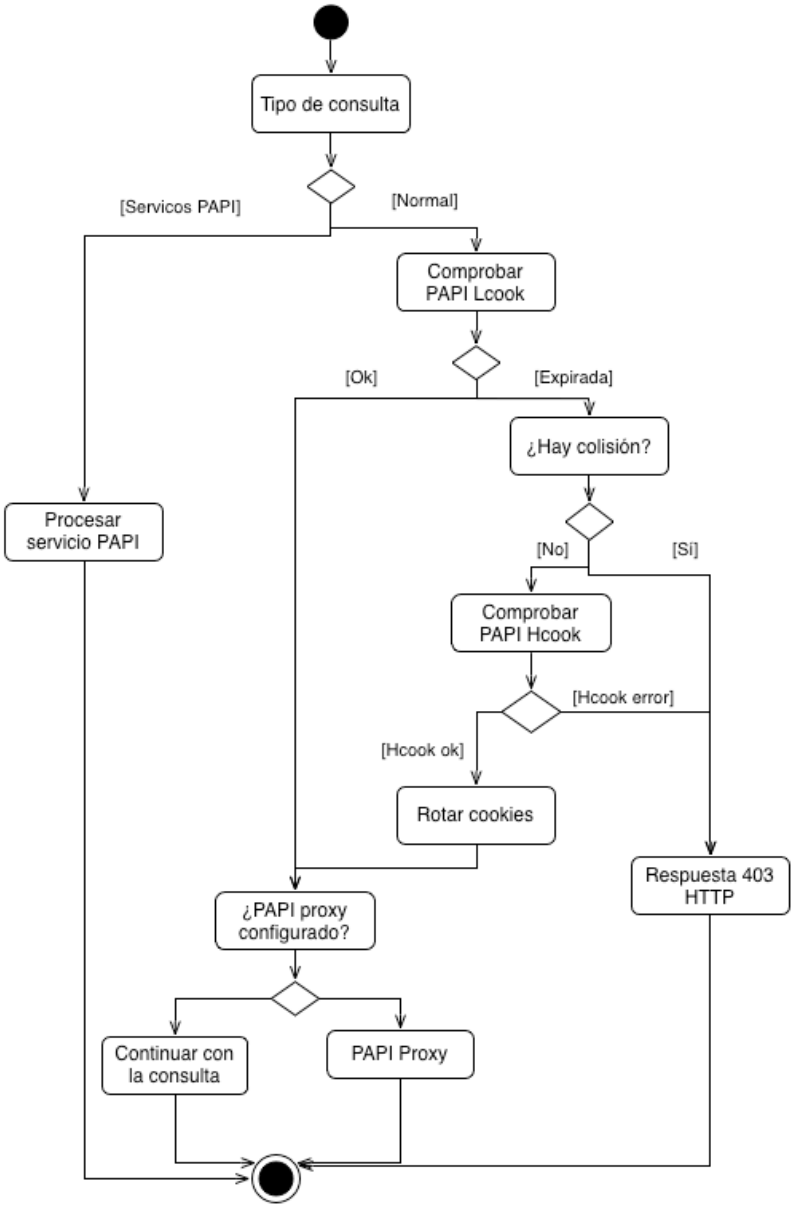

Fig. 2-4 *Diagrama de actividad del gestor principal en un punto de acceso PAPI*

Módulo gestor de cookies: Este módulo es el encargado de realizar todas las tareas relativas a las "cookies" PAPI. Entre ellas se pueden destacar:

- Rotación de cookies: Cuando las "cookies" PAPI expiran, éste módulo se encarga de comprobar la validez de la "cookie" "Hcook", y si todo es correcto, genera y registra en el repositorio de "cookies" PAPI, las nuevas "cookies "para ser enviadas al usuario. La utilización de las "cookies" se detalla en el punto 2.3.2, referente al protocolo PAPI. Las comprobaciones que realiza el módulo son:

- Comprobar si el cifrado es correcto. Para ello utiliza el algoritmo de clave simétrica correspondiente, actualmente AES [11] (Advanced Encryption Standard) de 192 bits.
- Comprobar que la "Hcook" se corresponde a la que está en vigor y no una ya expirada, en cuyo caso se genera una alarma de colisión. Es decir, se detecta que la "cookie" está siendo utilizada por más de un cliente, es decir que se ha realizado copia de "cookies" entre equipos. Esta comprobación se realiza verificando el campo aleatorio de la "cookie" a chequear con el campo aleatorio de la "cookie" en vigor.
- Comprobar que la fecha y hora de sesión PAPI (incluida dentro de la "cookie") es correcta y todavía no ha expirado.

• Administración de "cookies": Este módulo es responsable de procesar las consultas referentes a la creación o destrucción de "cookies" PAPI, que se corresponden con fases de comienzo y finalización de sesión ("Logging" y "Logout"). En el caso de la fase de inicio de sesión el PoA recibe una petición de creación de cookies PAPI desde el servidor de autenticación del usuario. Dicha petición está firmada por el SA y por tanto el módulo gestor de cookies comprueba la firma utilizando la clave pública del SA cuya firma hay que comprobar. Si todo es correcto, genera nuevas cookies, las registra en el repositorio de Hcooks" en vigor, y se las pasa al gestor principal para que las incluya en la respuesta HTTP que se le envía al usuario. En el caso de la fase de cierre de sesión, el PoA recibe la consulta correspondiente, y es el modulo gestor de cookies el que, tras comprobar que este usuario tiene una sesión activa, genera cookies PAPI expiradas para que al ser incluidas en la respuesta HTTP al usuario, hacen que se eliminen dichas cookies del repositorio de cookies de la aplicación cliente, dando la sesión por finalizada.

Módulo de "proxy" inverso

Este módulo aunque queda fuera del ámbito del control de acceso puro, añade al sistema PAPI un gran potencial, ya que le permite actuar como interfaz de acceso de servicios remotos que funcionan en equipos diferentes a aquel en el que funciona el punto de acceso. Con ello se consigue que los servicios y recursos web puedan ser protegidos sin necesidad de modificar su configuración, y preservando al máximo su entorno, es decir, sin necesidad de tocar su configuración o instalación.

El módulo de "proxy" inverso tras recibir una consulta, analiza la URL (Uniform Resource Locator) de destino e inicia una nueva consulta utilizando la librería "libwww-perl" [12] como agente web a dicha URL. Finalmente, la respuesta recibida es comunicada al gestor principal del PoA para que devuelva el resultado al usuario. Si la respuesta a devolver es del tipo MIME (Multipurpose Internet Mail Extensions) "text/html", el módulo espera a completar la consulta para enviar la respuesta al usuario, mientras que en caso contrario, y con vista a grandes flujos de datos, el módulo está optimizado para ir devolviendo al usuario los datos según van siendo recibidos, lo que evita problemas de desbordamiento de memoria en el sistema.

Con respecto al sistema de "proxy" inverso empleado, el de PAPI utiliza tecnología de reescritura de URLs, de tal forma que al recibir una página web de respuesta del servidor remoto, y antes de ser entregada al usuario, el módulo reescribe las URLs de tal forma que fuerza al usuario a volver a pasar por el "proxy", lo que garantiza la accesibilidad al recurso a través del PoA. Este sistema de reescritura ha sido refinado a lo largo de los años y actualmente dispone de un rico juego de opciones que permiten, por ejemplo, tratar casos de URLs incluidas en "javascript" o en objetos de tipos MIME diferentes a "text/html".

2.3. El protocolo del sistema PAPI

Para la implementación del sistema PAPI en un primer momento se optó por utilizar certificados de usuario, pero rápidamente aparecieron problemas relacionados con: la usabilidad de la solución, la complejidad en la gestión de una autoridad de certificación, y la complejidad en la revocación de certificados. Por consiguiente, se desechó ese camino y se optó por un elemento que resulta mucho más transparente y fácil de utilizar para los usuarios, las “cookies”. Esta tecnología eliminaba una parte de los problemas, pero introducía otro más técnico. El problema se deriva de la restricción que sólo permite recibir las cookies aquel servidor que las originó previamente. Para solucionar este problema se forzó a que el sistema hiciera exactamente lo requerido, es decir, que fueran los PoAs los que entregaran las claves temporales que después ellos mismos utilizarían.

El protocolo que emplea el sistema PAPI consta de dos fases: autenticación y control de acceso. La fase de autenticación comienza cada vez que el usuario accede al

SA para conseguir las claves temporales. Una vez las ha conseguido, no debería volver a esta fase durante el tiempo en que sus claves son válidas. Sólo en ciertos casos el usuario tiene que volver a autenticarse de manera prematura:

- Cuando las claves temporales son borradas del navegador (en la implementación actual, cuando las cookies son borradas del fichero de cookies del navegador).
- Cuando las claves temporales se corrompen.
- Cuando las claves temporales son copiadas a otro equipo y el sistema detecta colisión entre dos usuarios utilizando las mismas claves.
- Cuando las claves de encriptación de cookies son cambiadas en algún PoA. Este caso se puede dar, por ejemplo, porque se sospecha que la seguridad ha sido comprometida, o también por política de seguridad que obliga cambiar las claves cada cierto tiempo.

2.3.1. La fase de autenticación

Esta fase comienza cuando el usuario se autentica y finaliza cuando las cookies temporales que le permiten pasar por los diferentes PoAs están almacenadas en el navegador. En la Fig. 2-5 se muestra un diagrama de secuencia de esta fase del protocolo:

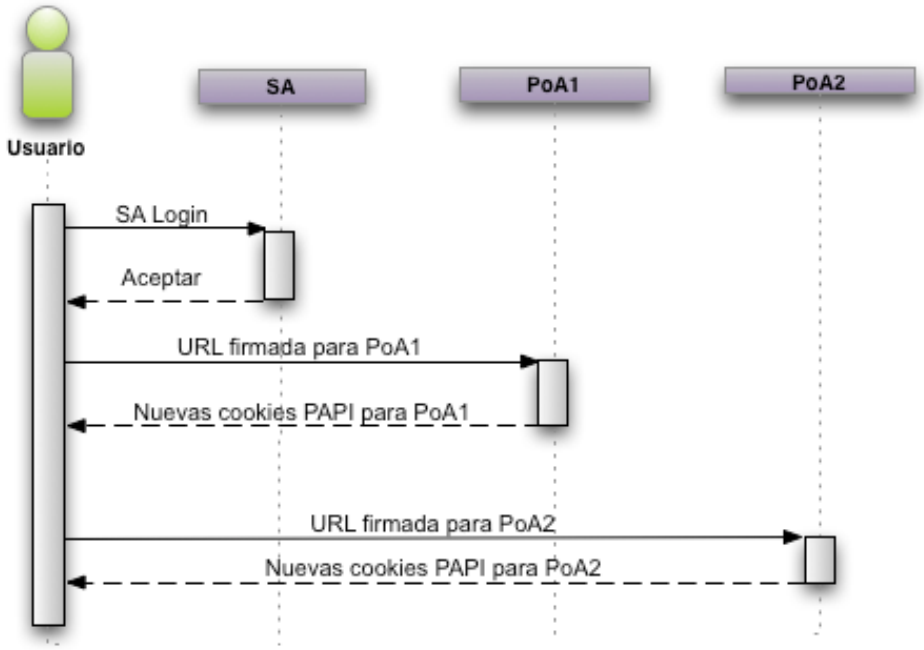

Fig. 2-5 *Diagrama de secuencia del protocolo PAPI en la fase de autenticación*

1. El usuario se conecta con el navegador al servidor de autenticación y le facilita la información que fuera necesaria para su autenticación. Esta información es analizada por el módulo de autenticación que la organización haya considerado oportuno, o que incluso haya desarrollado.
2. Si la autenticación es válida, el servidor consulta a qué recursos tiene acceso el usuario y durante cuánto tiempo. Esto puede tener relación con: qué usuario es, a qué departamento pertenece, desde dónde esta realizando la autenticación, etc. Una vez obtenida la lista de recursos a los cuales puede tener acceso este usuario, el SA genera una página web de respuesta con una serie de URLs firmadas (que no pueden ser modificadas o generadas por ningún elemento excepto por el servidor de autenticación). Las URLs firmadas tienen los siguiente campos:
   - **AS**: Nombre del servidor de autenticación
   - **ACTION**: Acción a realizar:
     - LOGIN: Petición de nuevas claves.
     - TEST: Comprobación de acceso a ese PoA.
     - LOGOUT: Desconexión, es decir, borrado de las claves de acceso.
   - **DATA**: Información sobre la conexión:
     - Fecha de creación de la URL.

- Fecha de expiración para el acceso.
- Código del usuario y aserción con atributos del mismo según la configuración del servidor de autenticación.

3. El navegador recibe esta página y resuelve las URLs antes descritas de una manera completamente transparente para el usuario, ya que cada una de ellas se corresponde con la carga de un gráfico o un elemento HTML (HyperText Markup Language) de similares características. Cada una de estas URLs se corresponde con un PoA del cual se quiere obtener su acceso.
4. Cuando un PoA recibe una URL firmada, éste comprueba que la URL ha sido generada por un SA válido y sus datos no han sido modificados.
5. Si es así, genera un par de cookies ("Hcook" y "Lcook") cifradas con las claves simétricas propias (como ya se ha comentado anteriormente) y las devuelve al navegador para que las guarde. La función de estas "cookies" se explica en detalle en el punto 2.3.2. La estructura de las cookies es la siguiente:
   - **Hcook**:
     - Fecha de expiración del acceso.
     - Código del usuario.
     - SA donde se autenticó.
     - Servidor para la cual es válida la "cookie".
     - "Location" para el cual es válida la "cookie".
     - Campo aleatorio de seguridad que se cambia cada vez que expira "Lcook"
   - **Lcook**:
     - Fecha de creación
     - Servidor para la cual es válida la "cookie"
     - "Location" para el cual es válida la "cookie".

A partir de este momento el usuario tendrá acceso a los recursos para los cuales ha obtenido las cookies correspondientes.

2.3.2. La fase de control de acceso

En esta fase, véase la Fig. 2-6, el usuario intenta acceder a recursos, que están controlados por PoAs y a los cuales él tiene acceso.

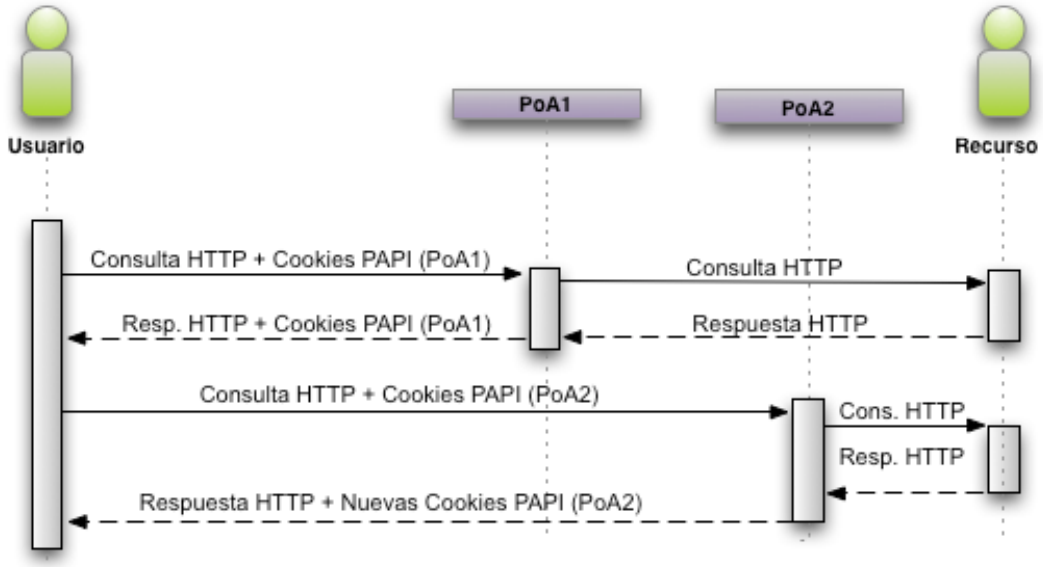

Fig. 2-6 *Diagrama de secuencia del protocolo PAPI en la fase de autorización*

1. Al realizar la consulta, el navegador envía automáticamente las "cookies" ("Lcook" y "Hcook") relativas a ese PoA. "Lcook" también lleva un tiempo de expiración corto que conlleva la renovación de "Hcook". Con ello se evita la copia de "Hcook" en varios equipos.
2. Cuando el PoA recibe la consulta, extrae las cookies y comprueba la validez de la "cookie" "Lcook". Si es válida deja pasar la consulta.
    a. Si no es válida, o ha expirado, comprueba "Hcook":
    b. Comprueba la fecha de expiración.
    c. Comprueba si este "Hcook" es el mismo que está registrado en la base local de "cookies" "Hcook".
    d. Comprueba si existe algún filtro para este usuario.
3. Si es válida, genera nuevas "Hcook" y "Lcook", registra la nueva "Hcook" como válida en la base local de "Hcooks" y deja pasar la consulta.

La utilización de dos cookies en vez de una sirve para dar una mayor eficiencia al sistema, sin perder seguridad. La clave "Lcook" lleva un sistema de encriptación más ligero, lo que permite una mayor rapidez en su chequeo. La clave "Hcook" lleva más información y una clave de encriptación más fuerte y más pesada a la hora de ser

comprobada. Esta comprobación se realiza cada más tiempo, con lo cual se consigue no perder eficiencia, aunque se sigue manteniendo la robustez de una buena encriptación.

Capítulo 3

Implantación de PAPI como infraestructura de seguridad para las aplicaciones del TJ-II

3.1. Ámbito del problema

3.1.1. Los sistemas del TJ-II.

El TJ-II es un dispositivo de fusión por confinamiento magnético de tamaño medio de tipo "stellarator" situado en el CIEMAT. Este dispositivo funciona en modo pulsado, es decir su actividad se organiza en pulsos de una duración aproximada de 500 ms. Tras cada pulso, se reconfigura todo el sistema para el siguiente pulso y así sucesivamente. Uno de los sistemas esenciales asociados al TJ-II es el sistema de adquisición de datos [13] [Referencia], responsable de dotar a los investigadores e ingenieros de los recursos necesarios para una correcta adquisición y análisis de los datos: gestión y control de los canales de digitalización [14,15,16], gestión y control de los procesos de adquisición [13], gestión y acceso a los datos experimentales [17,18,19] así como la integración de datos experimentales procedentes de otros sistemas de análisis o procesamiento [16].

El ciclo de un pulso consiste en una serie de fases, como muestra la Fig. 3-1. Existe una fase de puesta a punto del sistema donde se configuran los equipos y se establecen los parámetros de adquisición (como por ejemplo la frecuencia de muestreo o las muestras que se toman por canal). A continuación, durante la descarga, los canales de adquisición muestrean y codifican los valores producidos por los equipos de diagnóstico y generan señales de evolución temporal, cuyos datos son almacenados temporalmente en almacenes temporales locales. Al finalizar la descarga, los canales envían los datos

almacenados en sus almacenes temporales locales al sistema de compresión y almacenamiento de la información. Este sistema es el encargado de unificar toda la información relativa al pulso en una estructura de datos llamada "fichero de descarga" [20], y en un segundo nivel, se encarga de comprimir la información. La técnica de compresión utilizada [21] permite recuperar los datos sin ningún tipo de distorsión espectral y en el caso del TJ-II, la tasa de compresión media es superior al 70%. Una vez creado el fichero de descarga es posible realizar los procesos de análisis sobre los datos obtenidos.

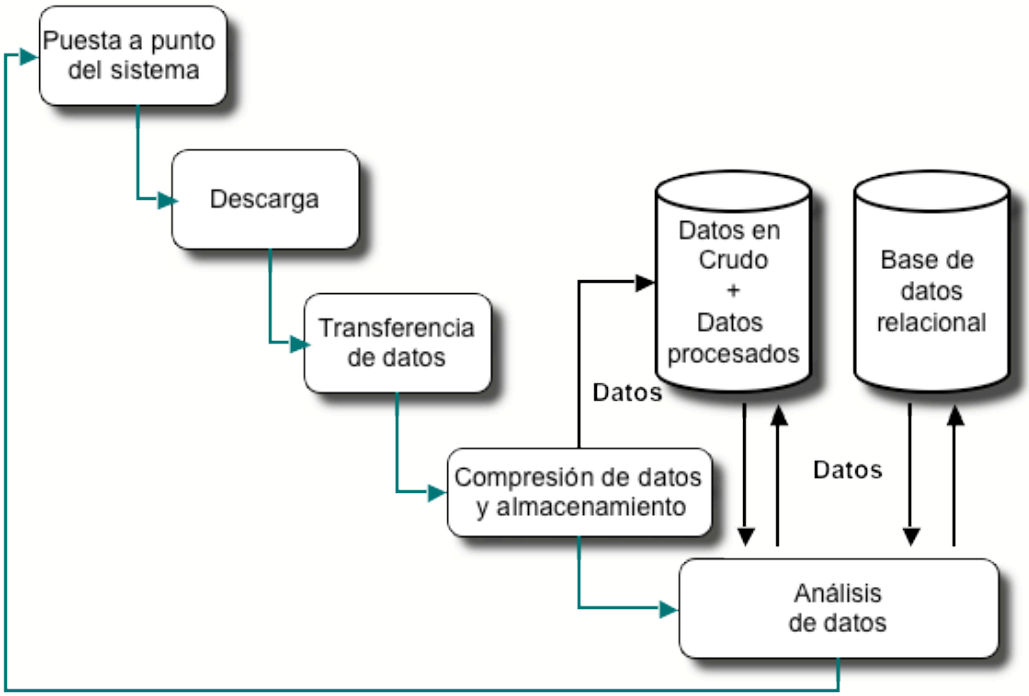

Fig. 3-1 Ciclo de un pulso del sistema TJ-II

El núcleo del sistema de adquisición TJ-II se estructura en una serie de equipos de digitalización conectados mediante una red de área local a un grupo de servidores UNIX centrales encargados de gestionar y almacenar los ficheros de descarga, y en los que han implementado una serie de métodos de acceso a los datos basados en librerías RPC (Remote Procedure Call). Además de estos elementos existen bases de datos relacionales donde se almacenan y gestionan datos relativos a la configuración de los equipos de adquisición, y a la operativa del TJ-II. El sistema de adquisición del TJ-II tiene una arquitectura distribuida y para la sincronización de sus equipos se implementó

el sistema de sincronización por software AEDS (Asynchronous Event Distribution System) [22], basado en la distribución de mensajes de sincronización utilizando la red de área local.

Desde el punto de vista de los usuarios, la generalización en el acceso a Internet, la proliferación de las redes WIFI, y los desplazamientos debido a requerimientos laborales, ha hecho que la movilidad se convierta en un parámetro prioritario desde el punto de vista del seguimiento de la operación del TJ-II. De ahí, que el sistema de participación remota del TJ-II fuera diseñado con el objetivo de poder ser utilizado indistintamente desde Internet o desde la red de área local del TJ-II, y así poder dar servicio a usuarios móviles.

3.1.2. El entorno de participación remota del TJ-II

Desde el principio, el TJ-II ha dispuesto de un conjunto de herramientas para el control, gestión y monitorización, tanto de su operativa, como de la adquisición de datos o el control de diagnósticos. Todas estas herramientas fueros desarrolladas de forma independiente y son accesibles desde diferentes puestos de la red. El sistema de participación remota del TJ-II [23] fue diseñado para que sirviera como punto único de acceso a todas esas funcionalidades, es decir, un entorno único que englobara diferentes herramientas:

Servicios generales: engloba aplicaciones y servicios comunes para los usuarios pertenecientes al laboratorio de fusión.

Room Booking System: Permite realizar reservas de salas de reunión y videoconferencia. El sistema de reserva está desarrollado sobre tecnología PHP y sus datos son almacenados en una base de datos MySQL.

Seguimiento de la operación: engloba a un conjunto de recursos que permite a un usuario monitorizar diferentes aspectos relacionados con la operación del TJ-II tales como la configuración y monitorización de los canales de adquisición y de los sistemas de control de diagnósticos así como el seguimiento del histórico de las descargas. Es importante enfatizar que la operación del TJ-II se ha comandado de forma remota desde una sala de videoconferencias en Cadarache [24].

Diario de operación: servicio electrónico para el registro de incidencias y comentarios sobre las descargas que se van sucediendo en cada campaña experimental.

Los usuarios que acceden a este servicio con rol de piloto pueden realizar entradas en el diario mientras que el resto de usuarios sólo tienen permisos de lectura.

Sistemas de adquisición de datos: Conjunto de procesos implementados con VxWorks y LabView que controlan diferentes sistemas de adquisición. La interacción con los procesos de LabView se realiza a través de un interfaz datasocket Virtual Instruments [25] mediante URLs con consultas tipo GET y los parámetros asociados. La comunicación con procesos VxWorks se hace con "sockets".

Aplicación de configuración de hardware: Integra los diferentes tipos de sistemas de adquisición. Mediante una interfaz de gestión, se puede definir el propietario de un módulo de adquisición, el sistema y ranura dónde está instalado además del código que permitirá la identificación del módulo en el resto de sistemas.

Aplicación de detección de hardware: Permite gestionar los módulos instalados en un sistema de adquisición.

Aplicación para asignación de señal: Permite gestionar la asignación de señales experimentales a canales de adquisición. La idea es dotar de una herramienta de fácil utilización que permita una rápida reconfiguración de canales y señales asociadas.

Aplicación de control de adquisición: Permite configurar los parámetros de adquisición de los módulos de adquisición. En la aplicación se integran las diferentes arquitecturas de adquisición, mostrando diferentes pantallas de configuración para cada una de ellas.

Aplicación de visualización de señales: Permite la visualización de diferentes señales, permitiendo al usuario configurar y personalizar diferentes aspectos relacionados con la visualización de señales.

Aplicación de búsqueda de patrones: Permite la búsqueda de señales en función de patrones similares.

Aplicación Eboard: Permite la gestión de diferentes aspectos de la operación.

Programación y monitorización de los sistemas de control de diagnósticos: Son sistemas controlados mediante sistemas LabView que implementan una interfaz HTTP para que las pantallas de control puedan ser visualizadas a través de un navegador web. Las acciones sobre los diferentes controles se realizan a través de consultas HTTP tipo GET que incluyen como parámetros la acción a realizar y valores asociados.

3.1.3. Arquitectura del entorno de participación remota

El entorno de participación del TJ-II esta basado, como muestra la Fig. 3-2 en arquitectura multicapa de tres niveles [26]: capa cliente, capa intermedia, y finalmente la capa de datos y sistemas base. El objetivo de utilizar este tipo de arquitectura es, por un lado, que las modificaciones que se tengan que realizar sobre los sistemas o componentes de una de las capas no afecten (o afecten lo menos posible) a los del resto de las capas, y por otro lado al desacoplar los sistemas de diferentes capas, se pueden utilizar diferentes soluciones en cada una de ellas sin perjuicio del resto. Por ejemplo, se pueden utilizar diferentes tipos de aplicaciones cliente, y los datos se pueden almacenar en diferentes sistemas de bases de datos, sin necesidad de tener que cambiar de forma significativa el resto de los elementos de la arquitectura.

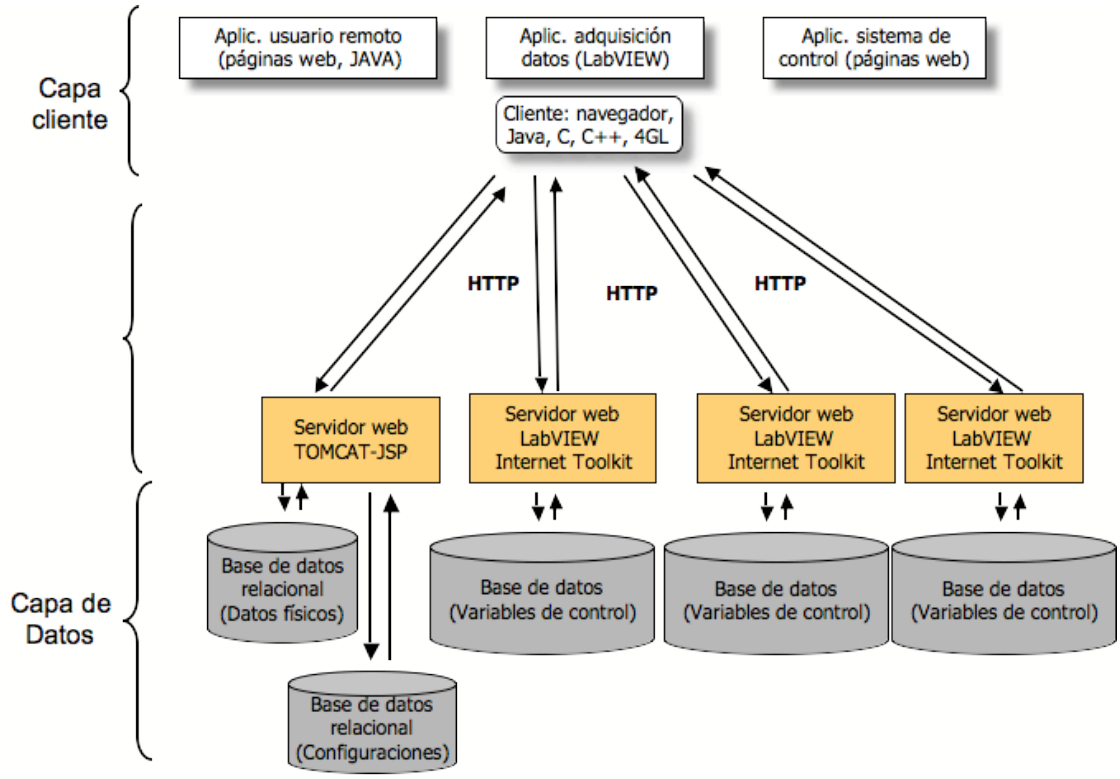

Fig. 3-2 *Arquitectura del entorno de participación remota del TJ-II*

3.1.3.1. La capa cliente

Las aplicaciones cliente utilizadas para acceder al sistema de participación remota del TJ-II se pueden dividir en dos grupos: navegadores web y aplicaciones JAVA a medida. El primer grupo esta formado por los navegadores más extendidos: Microsoft Internet Explorer, Firefox, Opera y Safari. En estos la interfaz de usuario se implementa principalmente mediante páginas web que incluyen código JavaScript para dotarlas de mayor capacidad de interacción con los usuarios.

El segundo grupo lo forman las diferentes aplicaciones JAVA desarrolladas a medida y que mantienen HTTP como protocolo base de comunicación. Esto hace que sus interacciones con los sistemas de la capa intermedia sean completamente compatibles con las de un navegador web, a la vez que su funcionamiento con el usuario mantiene todo el potencial de una aplicación JAVA. Estas aplicaciones utilizan librerías de conexión HTTP, y las más comúnmente utilizadas son: las clases estandar "java.net.*", la librería "commons-http" del proyecto Jakarta [27], y la librería HTTPClient de Ronald Tschalär [28].

Entre estas aplicaciones se encuentran la pizarra de operación (Eboard), la aplicación de visualización de señales, y la aplicación de búsqueda se señales. Estas aplicaciones utilizan el sistema de gestión de versiones JWS (Java Web Start) que permite, tanto el arranque de aplicaciones de una forma fácil para el usuario y con todas las opciones de ejecución incluidas, como una actualización de la aplicación de forma automática y completamente transparente para el usuario. La tecnología JWS (que será descrita en detalle en el punto 3.8.1) se basa en ficheros JNLP [29] (JAVA Network Launching Protocol), en los que se describe, en base a un formato XML (Extensible Markup Language) , diferentes aspectos relacionados con el entorno de ejecución de la aplicación asociada.

3.1.3.2. La capa intermedia

Capa intermedia entre las aplicaciones cliente, y la capa de datos y sistemas base. En esta capa se implementan una serie de aplicaciones web que interactúan a través del protocolo HTTP, lo que estandariza el protocolo de acceso para todos los servicios,

facilita la el acceso en remoto a través de cortafuegos y mejora la mantenibilidad de las aplicaciones.

La capa está compuesta por dos grupos de servicios. Por un lado, sistemas de control y monitorización "LabView" de los diferentes subsistemas de diagnóstico del TJ-II. Por otro lado, servidores de aplicación "Tomcat" [30] que proporcionan el resto de funcionalidades del sistema. Como ya se ha comentado, todos estos sistemas utilizan HTTP como protocolo de acceso a sus servicios, lo que permite a las aplicaciones acceder a todas las funcionalidades integradas en el TJ-II a través de consultas HTTP estándar. Además, es posible utilizar el navegador web como herramienta base de los usuarios, junto con la integración de tecnologías tipo JWS como sistema de acceso a las aplicaciones JAVA desarrolladas.

3.2. Comparación entre PAPI, Radius y Kerberos

3.2.1. Radius

El servicio RADIUS [7] (Remote Authentication Dial In Service) comenzó siendo un simple servidor de autenticación, orientado a autenticar las conexiones a red, y hoy en día es un protocolo estándar de la IETF (Internet Engineering Task Force) para autenticación, autorización [31] y registro [32]. Los servidores RADIUS son ampliamente utilizados hoy en día, sobre todo en sistemas relacionados con el acceso a redes y conectividad: DSL (Digital Subscriber Line), WIFI, VPNs (Virtual Private Network), puertos de conexión Ethernet, etc.

Los elementos que interaccionan en un sistema RADIUS son tres. En primer lugar, el cliente que es autenticado y que puede requerir el uso de ciertos servicios. En segundo lugar, el NAS (Network Access Service), que es el elemento al cual el cliente solicita el servicio y que utiliza un servidor RADIUS para saber si el cliente está autorizado para obtener dicho servicio. Finalmente, el servidor RADIUS, que es el que se encarga de aceptar o rechazar la solicitud del cliente en función de la información que recibe del propio cliente o de reglas de acceso e información que él almacena bien localmente, bien en repositorios asociados. El servidor RADIUS puede realizar consultas a estos repositorios para completar información relativa al cliente, e incluso, puede delegar la decisión de autenticación a servidores externos.

3.2.1.1. Protocolo RADIUS

El protocolo se define a nivel de aplicación y se basa en el intercambio de paquetes UDP (User Datagram Protocol). Se puede dividir en dos fases, primeramente la de autenticación y autorización y segunda la fase de registro. Para cada una de ellas, el estándar define una serie de mensajes, junto con su formato, que son intercambiados entre el cliente, el NAS y el servidor RADIUS. Como se muestra en la Fig. 3-3 la fase de autenticación y autorización consta de los siguientes pasos:

1. El cliente solicita al NAS acceso a un servicio mediante un Access-Request.
2. El NAS reenvía la solicitud al servidor RADIUS
3. El servidor RADIUS responde con un Access-Accept (aceptando la petición), un Access-Reject (rechazando la petición), o bien un Access-Challenge , en el cual el servidor RADIUS solicita información adicional para poder tomar la decisión.

Para la fase de registro, el cliente y el servidor intercambian mensajes de tipo "Accounting-Request" ("start", "interim-update" o "stop") y "Accounting-Response".

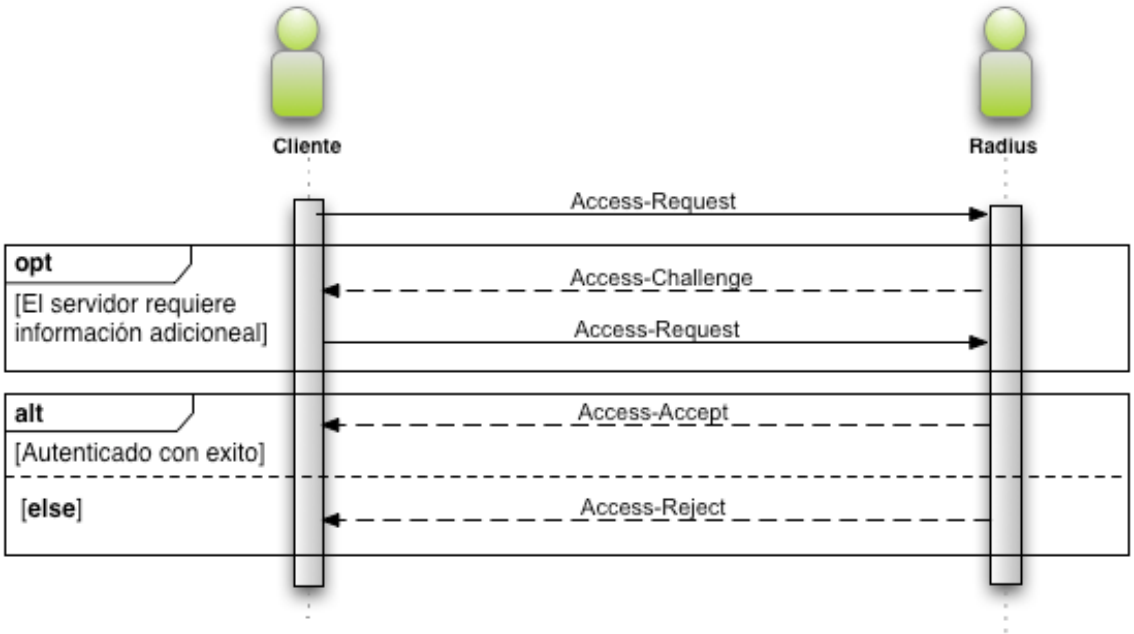

Fig. 3-3 *Diagrama de secuencia del protocolo de autenticación y autorización de "Radius"*

La estructura del mensaje RADIUS consta de una cabecera y un conjunto de atributos, cada uno de cuales incluye información a intercambiar entre clientes, NAS, y servidores RADIUS durante las fases de autenticación/autorización y registro. En los mensajes de tipo "Access-Request" se incluyen atributos relacionados con credenciales de usuario y propiedades del servicio solicitado, mientras que en los mensajes de tipo "Access-Accept" se incluyen atributos con información relativa al tipo de conexión o servicio autorizado (como la VLAN a utilizar) y atributos VSA ("vendor-specific attributes") específicos del servicio o del fabricante del equipo que provee el servicio o la conexión. Los atributos reconocidos en el protocolo RADIUS se definen en la configuración de los equipos implicados en el protocolo mediante diccionarios de atributos.

3.2.1.2. Autenticación y Autorización

Existen un conjunto de métodos de autenticación soportados en el protocolo RADIUS como son PAP, CHAP, MS-CHAP y EAP [33].

PAP ("Password Authentication Protocol") se basa en la transmisión en formato ASCII del nombre de usuario y clave secreta sin cifrar, por lo que se considera un método inseguro de autenticación.

CHAP (Challenge-Handshake Authentication Protocol) se basa en la transmisión de las credenciales del usuario, previa aplicación de una función "hash", cada vez que el servidor RADIUS le envía al cliente un mensaje "Access-Challenge". Este método requiere de clave compartida entre el cliente y el servidor.

MS-CHAP (Microsoft CHAP) es una extensión del método CHAP y añade compatibilidad con servicios y equipos Microsoft. Además posibilita que el servidor no necesite compartir clave con el cliente y añade mensajes de error. Existe una segunda versión MS-CHAP-v2 que incluye autenticación mutua entre cliente y servidor, y utiliza claves diferentes para envío y recepción, lo que le hace criptográficamente más fuerte que su antecesor.

Para posibilitar la inclusión de nuevos métodos de autenticación se habilito EAP [34,35] (Extensible Authentication Protocol) en el protocolo RADIUS. EAP establece un método de intercambio de mensajes entre el cliente y el servidor que permite la inclusión de nuevos métodos de autenticación entre el cliente y el servidor RADIUS sin

más que añadir en ambos extremos los plug-ins correspondientes. Gracias a EAP se han añadido a RADIUS métodos de autenticación como: "Generic Token Card", "One Time Password" y "MD5-Challenge". Aparte de estos nuevos métodos de autenticación, gracias a EAP se han podido incluir métodos de autenticación sobre canal cifrado SSL basados en tecnología de clave pública, lo que permite eliminar ataques de tipo "hombre en el medio" y asegura la comunicación entre el cliente y el servidor RADIUS independientemente de los equipos y servidores que haya por el medio. Los dos métodos de autenticación de este tipo más extendidos son EAP-TLS (EAP-Transport Layer Security) y PEAP (Protected Extensible Authentication Protocol). El primero de ellos crea un canal SSL basado en los certificados del cliente y del servidor RADIUS, mientras que el segundo sólo requiere de certificado de servidor para crear el canal SSL.

3.2.1.3. Multiorganización

Diferentes servidores RADIUS que pueden pertenecer a diferentes organizaciones pueden coordinarse de tal forma que un cliente puede solicitar un servicio en una organización externa y todo el proceso de autenticación/autorización se realiza contra el servidor RADIUS de su organización origen. Un ejemplo de este tipo de organización es "eduRoam" [36]. RADIUS se utiliza comúnmente como método de control de acceso para conexión WIFI y las organizaciones tienen sus puntos de acceso WIFI registrados en un servidor RADIUS central. La iniciativa "eduRoam" coordina diferentes organizaciones para que un usuario, como se muestra en la Fig. 3-4 cuando visita otra organización, pueda obtener conexión WIFI utilizando credenciales reconocibles por el servidor RADIUS de su organización origen y con la seguridad de que ningún elemento intermediario pueda robar dichas credenciales. Para ello se utilizan, por un lado, la capacidad de los servidores RADIUS de reenviar mensajes a otros servidores. Por otro lado, la posibilidad de conocer la organización origen del usuario en base a su identificador. Por último se hace uso de la capacidad de protocolos como EAP-TLS y PEAP para poder crear canales cifrados entre el cliente y el servidor RADIUS final.

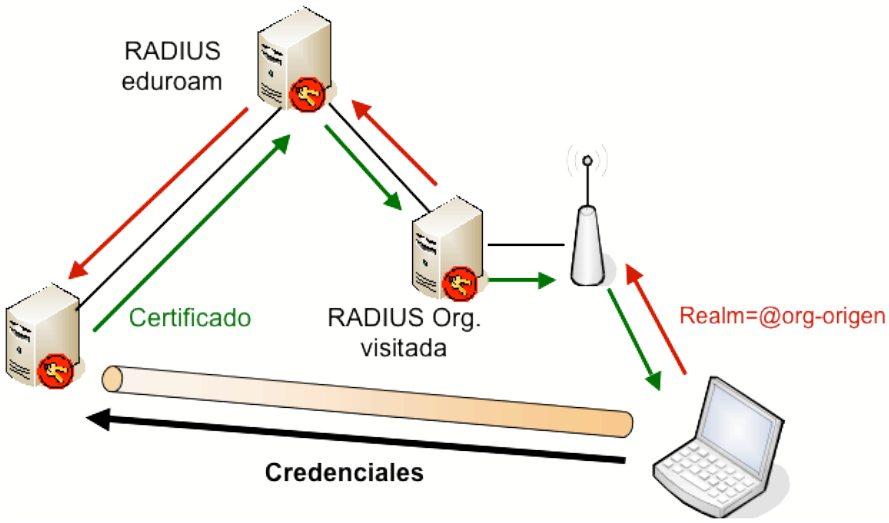

Fig. 3-4 *Esquema de una infraestructura "Radius" entre organizaciones*

3.2.2. Kerberos

"Kerberos" [37] es un protocolo de autenticación para equipos conectados en red que permite la autenticación segura entre dichos equipos, aunque la infraestructura de red sobre la que se comunican sea insegura. "Kerberos" fue desarrollado en la mitad de los años 80 en el MIT (Masachuset Institute of Technology). Desde entonces el sistema se ha extendido por todo el mundo como infraestructura de autenticación en red local y su protocolo se ha ido modificando para atender a las diferentes necesidades. Actualmente se encuentra oficialmente en la versión 5, y ha sido adaptado a la mayoría de los sistemas operativos. De hecho es el sistema de autenticación por omisión para dominios Windows a partir del Windows 2000.

3.2.2.1. El protocolo Kerberos

"Kerberos" se basa en el cifrado con clave simétrica. Aunque en un principio utilizaba el sistema de cifrado DES (Data Encryption Standard), posteriormente adoptó sistemas de cifrado más seguros como RC4 y AES. Estos sistemas de cifrado permiten la comunicación segura entre dos equipos que compartan una misma clave simétrica que sólo es conocida por ellos, ya que los mensajes cifrados con una clave sólo pueden ser leídos o modificados si se conoce dicha clave.

Aparte del sistema de cifrado, lo que caracteriza a "Kerberos" es la utilización de un servidor central KDC (Key Distribution Center), que actúa como tercera parte de confianza y que comparte claves simétricas con todos los equipos implicados en el protocolo. El KDC provee a los elementos que se quieren comunicar de "ticket" se sesión.

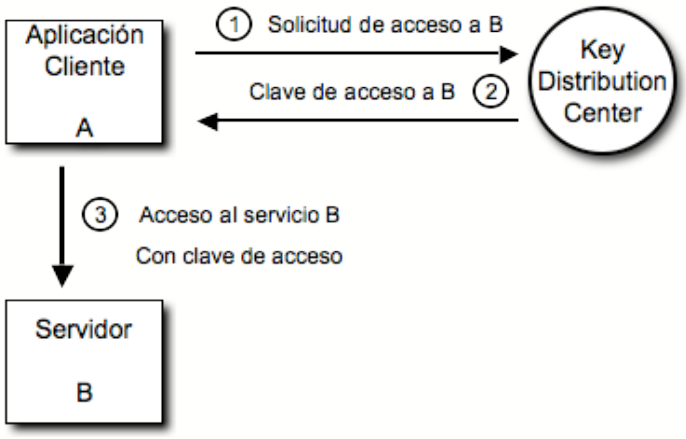

Fig. 3-5 *Diagrama de colaboración del protocolo básico Kerberos.*

Los "ticket" de sesión son un conjunto de informaciones que incluyen, entre otras cosas, el tiempo de validez del ticket y un identificador de sesión aleatorio. El ticket de sesión es convenientemente cifrado con las claves apropiadas para que los dos interlocutores que lo van a utilizar confíen en él sabiendo que sólo el KDC ha podido generarlo, que sus datos no han podido ser modificados y que sirve para una sesión de tiempo limitado entre los dos equipos para los que ha sido creado. Este "ticket" es incluido como parte de la conexión al servicio destino. Como se muestra en la Fig. 3-5 los pasos para la solicitud de un ticket de sesión son:

1- El cliente A solicita al KDC acceso al servicio B. En la solicitud se incluye: un identificador del cliente A, un identificador para el servicio B, un tiempo de expiración para la sesión y un código aleatorio.

2- El KDC le responde (en caso afirmativo) con un mensaje en 2 partes. Una primera que contiene: una clave de sesión $K_{A,B}$ (con la que cifrar las comunicaciones entre A y B) y el random enviado en la consulta, todo esto cifrado con la clave de A. Y

una segunda, para ser enviada a B junto con la solicitud de servicio, que contiene el "ticket" de sesión (que a su vez contiene la clave de sesión junto con su tiempo de expiración) cifrado con la clave de B.

3- Cuando A solicite el servicio a B, le enviará la consulta cifrada con la clave de sesión, junto el ticket de sesión cifrado con la clave de B. A continuación este extraerá la clave de sesión del "ticket" de sesión y podrá descifrar el contenido de la consulta.

Con el fin de reducir el riesgo de exposición de las claves básicas y para dotar al protocolo de mayor transparencia y flexibilidad, se incorpora el concepto del proveedor de "tickets" TGT (Ticket Granting Tickets). La idea es la de no exponer continuamente las claves básicas para la solicitud de "tickets" de sesión para la conexión de un cliente con un servicio y en su lugar utilizar la clave de sesión asociada a un TGT para obtener los posteriores "tickets" de sesión. En la Fig. 3-6 se muestran los pasos del protocolo mejorado:

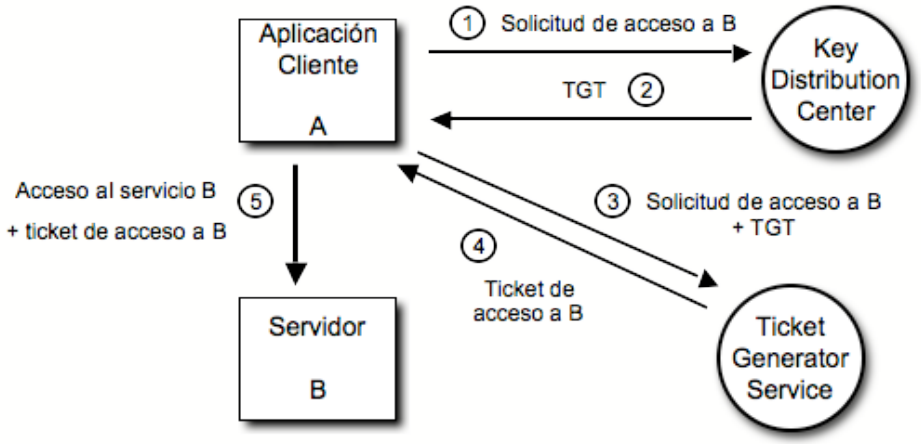

Fig. 3-6 *Diagrama de colaboración del protocolo kerberos utilizando "Ticket Grant Ticket"*

1- El cliente A solicita al KDC un TGT.

2- El KDC, si todo es correcto, le envía al cliente A un TGT que contiene: una clave de sesión y un "ticket" de sesión para ser utilizado con el servidor generador de "tickets" de sesión TGS (Ticket Generator Service).

3- El cliente A solicitará "tickets" de sesión (con su clave de sesión correspondiente) al TGS cada vez que quiera conectarse a un servicio. De esta forma, su

clave básica inicial queda preservada y es la clave de sesión asociada al TGT (que tiene una validez temporal) la que queda expuesta.

4- El TGS devuelve al cliente A una clave se sesión que podrá ser utilizada por éste, durante un tiempo limitado, para comunicarse con el servicio solicitado.

5- El cliente A utiliza el "ticket" de sesión como parte del mensaje a enviar al servicio B.

3.2.2.2. Extensiones al protocolo

Microsoft a partir del "Windows 2000" ha incorporado "Kerberos" como su sistema de autenticación en red por defecto. Además, ha incorporado mejoras al protocolo que permiten la utilización de tecnología de clave pública para la autenticación del usuario frente al KDC, como por ejemplo la autenticación basada en "Smart Card". El protocolo es el siguiente:

1- Al usuario se le requiere el PIN de su tarjeta "Smart Card", y con el se extrae el certificado X.509 que lleva dentro.

2- El cliente envía al KDC la petición inicial de autenticación en la que se incluye el certificado público del usuario.

3- El KDC chequea la validez del certificado y devuelve la clave de sesión cifrada con la clave pública del usuario, junto con un TGT.

4- El usuario que está en posesión de la clave privada correspondiente, podrá descifrar la clave de sesión. A partir de este paso el resto del protocolo es idéntico.

3.2.3. Certificados digitales

Las infraestructuras de seguridad basadas en certificados digitales resultan cada vez más comunes. El certificado digital es un documento en formato digital, con cierta información relevante sobre el objeto o servicio para el que fue creado, y que implementa una serie de mecanismos que permiten garantizar la integridad de todo el documento. La forma en la que se utilizan se asemeja mucho al uso cotidiano de otro tipo de documentos de identificación como el DNI (Documento Nacional de Identidad). En general, cuando un usuario requiere un servicio, anexa de cierta forma su certificado en la consulta, de tal forma que el servidor es capaz de identificar al usuario a través del

certificado "anexo" a la consulta. Además de la función de identificación que realiza el certificado digital, el servidor puede utilizar información incluida en el certificado como entrada a reglas de acceso que le permitan decidir si ese usuario tiene o no acceso al servicio. Asimismo, gracias a la tecnología subyacente al certificado digital, es posible establecer canales seguros de comunicación entre el cliente y el servidor.

Los métodos de cifrado utilizados en la tecnología de certificados digitales se basan en métodos de clave asimétrica, también denominados de clave pública. A diferencia de los algoritmos de cifrado de clave simétrica que utilizan una única clave para cifrar y descifrar datos digitales, los algoritmos de cifrado de clave asimétrica se basan en la utilización de un par de claves para poder cifrar y descifrar. Un mensaje cifrado con una de las claves, solo puede ser descifrado con la otra. Aunque desde el punto de vista del algoritmo no existe ninguna diferencia entre las claves (a nivel funcional), derivado de cómo son utilizadas estas claves en la implementación de sistemas reales se habla de una clave "pública" y una clave "privada". En entornos de seguridad basados en cifrado asimétrico, una de las claves (del par de claves) sólo puede ser visible para el elemento propietario de dicha clave, (a ésta se la llama clave privada), mientras que la otra clave es distribuida a varios elementos del sistema (y se la conoce como clave pública). De ahí la denominación de soluciones basadas en clave pública. En estos sistemas, como se muestra en la Fig. 3-7, la clave privada se utiliza para firmar mensajes, ya que, por un lado, sólo el poseedor de la clave privada puede haber cifrado el mensaje, y cualquiera que posea una de las copias de la clave pública podrá descifrar el mensaje, y por tanto, verificar la integridad de dicho mensaje. Como se puede apreciar el mecanismo se asemeja mucho al concepto de firma en documentos físicos. Por otro lado, la clave pública se utiliza para enviar mensajes cifrados "ocultos" al poseedor de la clave privada, ya que éste es el único que va a poder descifrar el contenido real del mensaje. Esta última funcionalidad permite el paso de una clave se sesión de cifrado simétrico, para poder crear un canal oculto entre dos elementos del sistema, como por ejemplo, entre un cliente y un servidor. Los algoritmos más comúnmente utilizados de clave asimétrica son: RSA desarrollado en el MIT, y DSA (Digital Signature Algorithm) desarrollado por en la agencia de seguridad nacional de estados unidos.

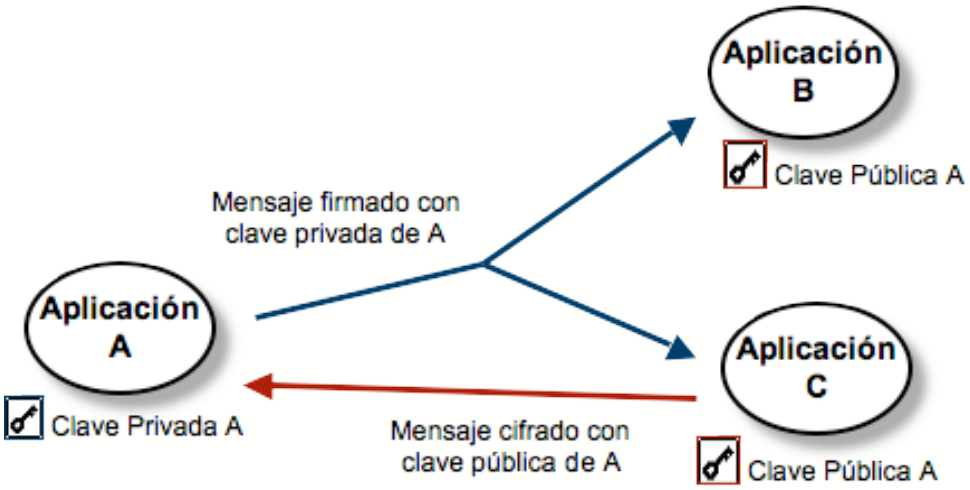

Fig. 3-7 *Esquema del uso de la clave asimétrica para firma y cifrado de mensajes*

Como ya se ha comentado, la tecnología principal de los certificados digitales es la firma digital, que es el mecanismo que permite, por un lado garantizar la integridad de los datos que forman un documento digital (en este caso en concreto de los datos del certificado), y por otro lado garantizar la identidad del firmante, y finalmente permite garantizar el no repudio de lo firmado.

Existen diversos métodos de firma digital, y en el caso de los certificados digitales, estos métodos se basan en algoritmos de clave pública. En la

Fig. 3-8 se representan los procesos de firma y verificación. El primero de ellos tiene los siguientes pasos:

1. Al documento se le aplica una función "Hash": estas funciones calculan un bloque de "bits" a partir del documento original, de tal forma que la probabilidad de que al aplicar la misma función "hash" a otro documento distinto se obtenga el mismo bloque de bits sea muy baja. A esta propiedad se llama fortaleza del algoritmo frente a colisiones. El algoritmo más comúnmente utilizado en certificados digitales y que no ha sido roto hasta ahora es el SHA-1.
2. Cifrado del bloque de datos hash: para el cifrado se utiliza un algoritmo de cifrado de clave pública. Los más comúnmente utilizados son RSA y DSA.

3. Al documento original se le anexa la firma y se convierte en un documento firmado.

Respecto al proceso de verificación sus pasos son:

1. A partir del documento firmado se obtiene el documento sin firmar y la firma.
2. Al documento sin firmar se le aplica la función "hash" y se obtiene el bloque de bits correspondiente a su "hash".
3. Por otro lado, la firma se descifra utilizando el algoritmo de clave pública correspondiente y la clave pública pareja a la que se usó para firmar el documento, con lo que se obtiene el "hash" del documento original.
4. Si los dos "hash" son iguales, la firma es correcta.

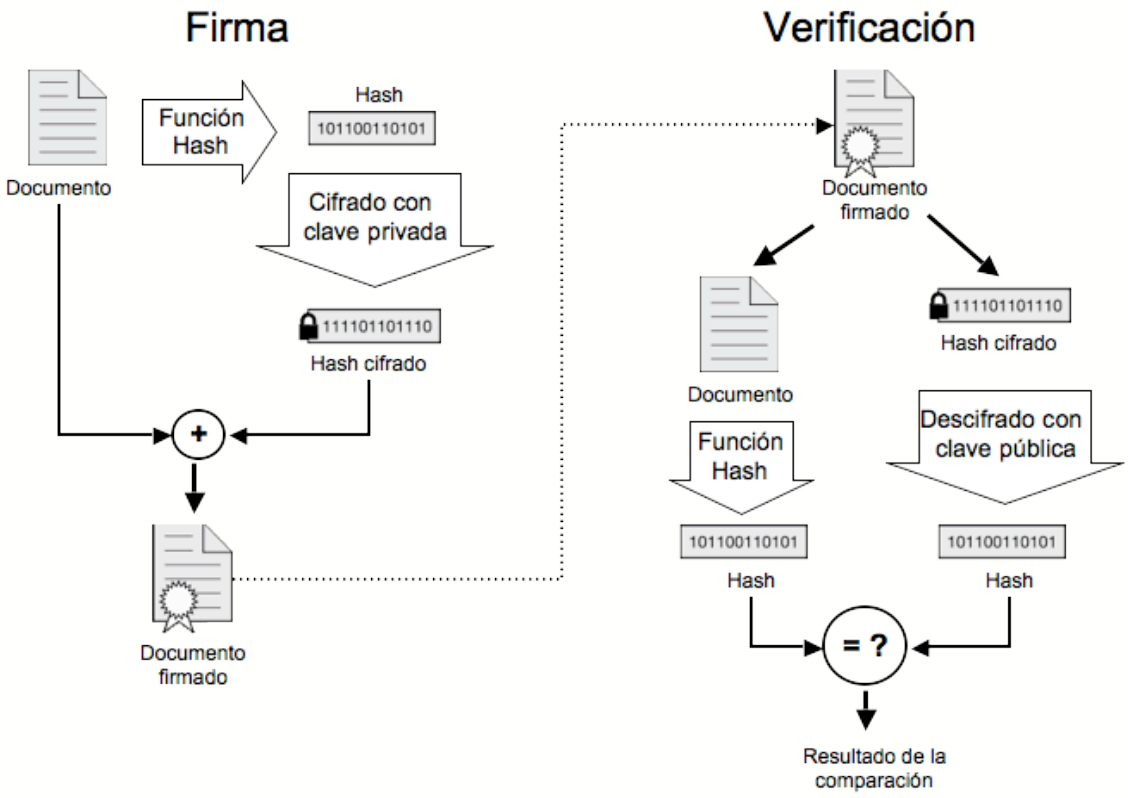

Fig. 3-8 *Procesos de firma y verificación de documentos digitales*

La implantación de una infraestructura de seguridad en base a certificados requiere de un mecanismo de confianza. Esto se puede obtener bien intercambiando las claves públicas de todos los individuos del sistema, bien utilizando autoridades de certificación. La autoridad de certificación es una entidad en la que confían, todos los elementos de un entorno y cuya función es firmar los certificados utilizados en dicho entorno. Previo a firmar un documento para convertirse en un certificado, una CA (Certification Authority) debe verificar si la información que contiene es correcta y veraz. A esta función se la llama registro y existen infraestructuras de autoridad de certificación en que dicha función es delegada a entidades llamadas "Autoridades de Registro". En estos casos la autoridad de certificación simplemente firma el certificado, dando por válida la comprobación realizada previamente por la autoridad de registro correspondiente. Otra propiedad de los esquemas de certificados basados en autoridades de certificación, es que éstas pueden organizarse de forma jerárquica. Es decir, una CA puede delegar en una o varias CAs con capacidad para firmar sus propios certificados. De esta forma un certificado puede estar firmado por una CA que a su vez puede estar firmada por otra, y así hasta la CA raíz que sólo esta firmada por ella misma. Es decir posee un certificado autofirmado. A esta estructura recursiva de firma se la llama cadena de certificación. A todo el sistema de gestión de certificados se le llama PKI (Public Key Infrastructure) [38].

Un certificado contiene básicamente datos relativos al propietario del certificado, y la clave pública asociada al certificado. Por supuesto a esta clave pública le corresponde otra privada que mantiene en secreto el propietario del certificado. Además, contiene información relativa al proceso de firma (entidad firmante, algoritmo de firma, etc.) e información relativa a su uso (periodo de validez, propósito, etc.).

En una infraestructura de seguridad basada en la utilización de certificados firmados por una autoridad de certificación, cuando un cliente se autentica frente a un servidor, le envía su certificado. El servidor comprueba la firma del certificado por parte de la autoridad de certificación común a ambos, y si todo es correcto, envía al cliente un código aleatorio cifrado con la clave pública incluida en el certificado. A partir de ese punto el cliente debe poder descifrar el código enviado por el servidor, lo cual significa que es realmente propietario de la clave privada asociada a la clave pública del certificado, y por tanto propietario del certificado presentado al servidor. Respecto a la autorización, el servidor puede aplicar ciertas reglas de control de acceso en base a la

información contenida en el certificado de cliente. Si no fuera suficiente, y se requiriera más datos sobre el usuario, están emergiendo soluciones basadas en certificados de atributos. Estos certificados contienen información adicional sobre el usuario, verificada y firmada por la autoridad de certificación competente. Normalmente las infraestructuras de certificados de atributo se implementan incluyendo índices a estos en los certificados "principales" (los que identifican al elemento). De esta forma si un servicio requiere información adicional sobre el elemento, puede a través de uno de estos índices, obtener el certificado de atributo asociado. Gracias a los certificados de atributo se pueden definir reglas de acceso más complejas.

3.2.4. Comparativa entre alternativas

Los cuatro sistemas han demostrado a lo largo de los años que llevan funcionando, su capacidad como sistemas de seguridad. En este punto no se va a discutir dicha capacidad, sino que se va a evaluar su nivel de adaptación y de eficacia al ser integrados como solución de control de acceso en el entorno de participación remota del TJ-II. Para realizar dicha evaluación se van a considerar una serie de aspectos relativos a sistemas de control de acceso y que son relevantes en el problema que se ha de resolver.

3.2.4.1. Gestión de usuarios

El esquema de gestión de usuarios utilizado tanto en "Radius" como en Kerberos, al ser soluciones centralizadas, se realiza en la misma organización propietaria de los servicios y gestora del sistema de seguridad. Por el contrario, tanto PAPI como una infraestructura de certificados con autoridad de certificación permiten, gracias a su carácter de sistema distribuido, delegar la gestión de los usuarios en las organizaciones origen de los mismos.

Un esquema centralizado resulta funcional cuando el número de usuarios es reducido y asumible para la organización gestora, pero en cuanto se habla de varias organizaciones con un número importante de usuarios, los sistemas centrales tienen problemas de escalado. Desde el punto de vista del registro y actualización de los datos de los usuarios, es mucho más lógico que dicha información sea gestionada por las organizaciones origen de los usuarios, que son quienes mejor conocen dicha información y tienen derechos legales para su gestión. Desde este punto de vista, las

soluciones distribuidas (PAPI y PKI) se adaptan mucho mejor a las necesidades del problema.

3.2.4.2. Mecanismos de autenticación

En este punto todos los sistemas incorporan diferentes sistemas de autenticación, salvo PKI que sólo requiere que el usuario disponga del certificado y clave privada asociada. Para este último caso existen soluciones que facilitan el almacenamiento y portabilidad del certificado por parte del usuario y evitan su robo a nivel telemático, tales como las "Smart Cards", que en general requieren la instalación de librerías y controladores en el equipo cliente donde van a ser utilizadas.

En el caso de PAPI, existen diversos métodos de autenticación ya implementados y permite la integración de prácticamente cualquier esquema de autenticación que sea compatible con un navegador web.

3.2.4.3. Aplicaciones cliente

"Radius", como sistema de control de acceso, está completamente orientado a los servicios y equipos de conexión de red, y en general, las aplicaciones cliente que lo implementan son aquellas cuya principal funcionalidad es la de proveer algún tipo de servicio de red que requiere autenticación. Por otro lado, Kerberos es un sistema de seguridad que ha sido integrado en muchas aplicaciones gracias a la distribución de librerías que implementaban los diferentes aspectos del protocolo.

A nivel de navegador web, tanto PAPI, como los certificados de usuario se integran perfectamente, y Kerberos ha sido integrado recientemente a través del protocolo SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism).

3.2.4.4. Servicios compatibles

"Radius" se ha integrado en la mayoría de elementos de conexión de red, tales como: conmutadores Ethernet, servidores VPN, o puntos de acceso WIFI.

Kerberos es compatible con una gran variedad de servicios, entre los que se incluyen, servidores web y servidores de aplicaciones tipo Tomcat.

El sistema de control de acceso basado en certificados de usuario está integrado en una gran variedad de servicios entre los que se incluyen servidores web y servidores de aplicación tipo Tomcat.

PAPI es compatible con servidores Apache y Tomcat. Aún así, su solución de "proxy" inverso, que le permite funcionar como interfaz de acceso de servicios HTTP, le capacita como sistema de control de acceso a un gran número de servidores web y servicios HTTP.

3.2.4.5. Movilidad de los usuarios

Por un lado "Radius" es utilizado habitualmente como mecanismo de autenticación en entorno local y con el equipo asignado al usuario. Por otro lado Kerberos requiere la utilización del equipo del usuario, ya que es donde está almacenada su clave Kerberos. Los certificados de usuario pueden son independientes de la IP del equipo, sin embargo su almacenamiento va asociado al equipo del usuario, y aunque se disponga de un elemento seguro para su almacenamiento tipo "Smart Card", normalmente se requieren drivers y software que no es habitual encontrar instalados. Finalmente, PAPI es completamente independiente de la IP y del equipo que utilice el usuario para acceder a los servicios.

3.2.4.6. Conclusiones

Uno de los requisitos más importantes del entorno de participación remota del TJ-II es la utilización del navegador web como aplicación cliente de autenticación y acceso a los servicios. Esta restricción implica descartar a "Radius" como solución. Asimismo Kerberos ha sido muy recientemente integrada en algunos navegadores, en últimas versiones, pero sigue siendo una solución incompatible para navegadores comúnmente utilizados en el momento de tomar la decisión sobre qué sistema aplicar. Existen sistemas donde además es necesario tener instaladas en el equipo las librerías Kerberos, lo que dificulta la movilidad de los usuarios y les obliga a usar su propio equipo.

Desde el punto de vista de la gestión de usuarios que pertenecen a diferentes organizaciones, las soluciones distribuidas tipo PAPI o certificados de usuario son claramente ventajosas. En este sentido, la puesta en marcha y mantenimiento de una

autoridad de certificación que englobe a varias organizaciones requiere un nivel de recursos a tener en cuenta, ya que hay que gestionar tareas de: registro, firma y revocación de certificados, y estar conforme con los requisitos legales correspondientes.

Cara a los servicios a proteger, son compatibles las soluciones basadas en Kerberos, PAPI y certificados de usuario. Como ventaja de PAPI y los certificados frente a Kerberos se puede destacar la mayor capacidad para poder definir reglas de acceso en base a atributos del usuario. En el caso de los certificados de usuario, dichos atributos deben restringirse a los que se incluyen en el certificado en el momento de su creación, mientras que PAPI aporta más flexibilidad pudiendo definir la aserción del usuario y los atributos que incluye en cualquier momento.

Desde el punto de vista del uso, la tecnología de certificados electrónicos requiere formación y apoyo técnico. Además, diferentes navegadores gestionan de diferente forma los certificados y no les resulta fácil a los usuarios entender las interfaces que les proporcionan. En definitiva, es una tecnología que requiere un alto nivel de soporte y que todavía no está madura a nivel de utilización.

3.3. Desarrollo de la solución

La integración de PAPI en el entorno de participación remota del TJ-II se divide en dos partes. Por un lado la instalación de un servidor de autenticación, con el objetivo de utilizar lo más posible los repositorios de usuarios ya existentes a la vez que minimizar el impacto sobre los modelos de datos existentes en dichos repositorios. Por otro lado se procederá a integrar la capa de control de acceso en los sistemas ya existentes de tal forma que se minimice las modificaciones a realizar en los mismos. La Fig. 3-9 muestra un esquema modular del sistema final, basado en el presentado en el punto 3.1.3 donde se ha descrito el entorno de participación remota.

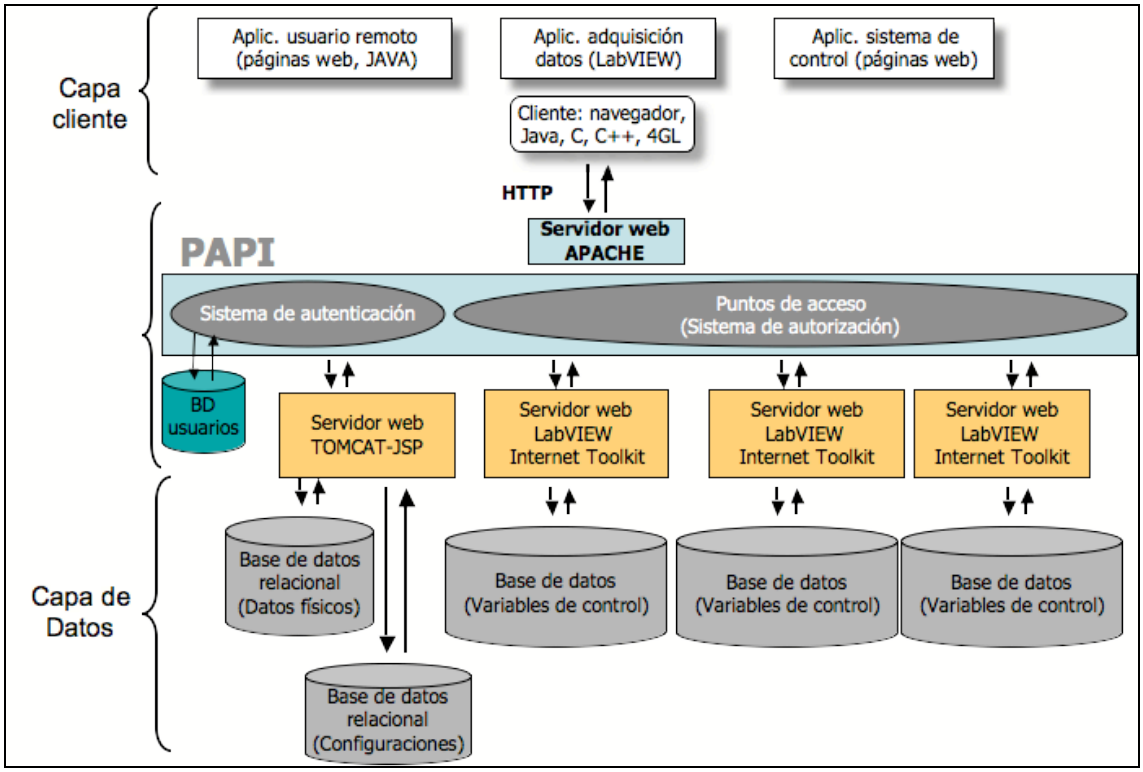

Fig. 3-9 *Arquitectura resultante de la integración de PAPI en el entorno de participación remota del TJ-II*

Tras una evaluación del entorno de participación remota del TJ-II, y en base al conjunto de soluciones a integrar en PAPI, se procede a la definición de una serie de tareas que marcarán el desarrollo del proyecto:

Integración del repositorio de usuarios: Es necesario extender el modelo de datos asociado al repositorio de datos con vistas a incluir la información que será utilizada por PAPI, tanto para la fase de autenticación como para las diversas reglas de control de acceso que se implementen.

Desarrollar un nuevo motor de control de acceso: Para poder dar respuesta a las políticas de control de acceso de algunos servicios, se requiere el desarrollo de un nuevo motor de autorización en el que se puedan definir expresiones lógicas y funciones de comparación, tanto para datos numéricos como para cadenas de texto, y en el que se tengan en cuenta, tanto atributos del usuario, como parámetros asociados a la consulta.

Integrar los servicios basados en aplicación de cliente JAVA: Para poder utilizar las distintas aplicaciones JAVA, se requiere la adaptación de ciertas librerías de tal forma que los accesos HTTP utilicen "cookies" (hay librerías que no manejan

adecuadamente las "cookies" en su implementación del protocolo HTTP), que las "cookies" se almacenen en un repositorio y que dicho repositorio sea común a todas las aplicaciones.

Integrar JWS: Como ya se ha comentado previamente, el entorno de participación remota del TJ-II utiliza JWS [29]como tecnología de distribución, ejecución automática y mantenimiento de sus aplicaciones JAVA. Esto más que un inconveniente, será aprovechado por PAPI como herramienta que permita la carga de credenciales y la utilización de las librerías adaptadas previamente comentadas.

3.4. Extensión del repositorio de usuarios

El repositorio de usuarios del laboratorio de fusión se divide en otros dos. El primero se ellos se trata de un directorio "Active Directory" en el que se almacenan los usuarios del laboratorio de fusión junto con el resto de usuarios del CIEMAT, y donde se almacena la contraseña de acceso a servicios centrales del CIEMAT como VPN o el servicio de acceso al correo electrónico vía web. El segundo repositorio es una base de datos relacional, implementada sobre "MS SQLServer", en donde se almacena información relevante para las aplicaciones y servicios del área de fusión.

El proceso de autenticación PAPI, una vez recibidas las credenciales, sigue los siguientes pasos:

1. El servidor de autenticación hace uso del módulo de identificación configurado, cuya función es comprobar si existe un usuario en el repositorio que se corresponde con las credenciales enviadas. De ser así, el módulo devuelve el identificador de usuario ("Uid") del repositorio para ser utilizado en las siguientes fases.
2. El servidor hace uso del módulo de obtención de los recursos y grupos de recursos para los cuales el usuario (y los grupos a los que pertenece) está autorizado.
3. Para cada recurso autorizado, el servidor construye la aserción de usuario que corresponda. La aserción es un campo texto que posee ítems del tipo '<papi attr="nombre del atributo"/>. Cada ítem es sustituido por el valor del atributo

que contiene. Dichos valores son recuperados previamente del repositorio de usuarios.

4. El servidor construye una lista de URLs firmadas, una por cada recurso autorizado, y que se utiliza como certificado electrónico a la hora de solicitar "cookies" al PoA asociado a dicho recurso o grupo de recursos respectivamente.

El repositorio de usuarios del que hace uso el servidor de autenticación requiere las siguientes entidades:

User: Donde se registran todos los usuarios reconocidos en el repositorio, junto con sus atributos.

Groups: Grupos de usuarios con su conjunto de atributos asociados.

Site: Esta entidad corresponde a los recursos o grupos de recursos a los cuales el usuario tiene acceso. Cada "site" se corresponde con un PoA o GPoA al cual el usuario tiene acceso. Como campos requeridos para esta entidad se encuentran la URL del PoA o GPoA asociado, el tiempo máximo de acceso al recurso antes de solicitar una nueva autenticación, las aplicaciones clientes requeridas por ese recurso (navegador web, aplicación JAVA o conexión con “token” de servicio), y finalmente el formato de aserción de usuario que se va a enviar en el acceso al recurso correspondiente.

URL: El conjunto de URLs asociadas a los recursos que el usuario tiene acceso y quieren ser mostradas tras una correcta autenticación.

En el caso del servidor de autenticación del CIEMAT, se optó por el sistema de base de datos SQLServer ya existente (como repositorio de usuarios) y el modelo entidad-relación que se implementó es el que se muestra en la Fig. 3-10. En él se pueden distinguir las entidades antes descritas y las relaciones entre cada una de ellas. Además, como módulo de identificación se implementó un proceso en dos fases. Una primera que sirve para consultar al directorio central del CIEMAT si las credenciales que manda el usuario ("Usuario" y "Contraseña") son correctas. Y una segunda fase para localizar el identificador de usuario (Uid) correspondiene al repositorio previamente descrito. Conviene remarcar que el usuario no es consciente de los procesos internos de la autenticación. Él se encarga de suministrar una única vez su "Usuario" y "Contraseña".

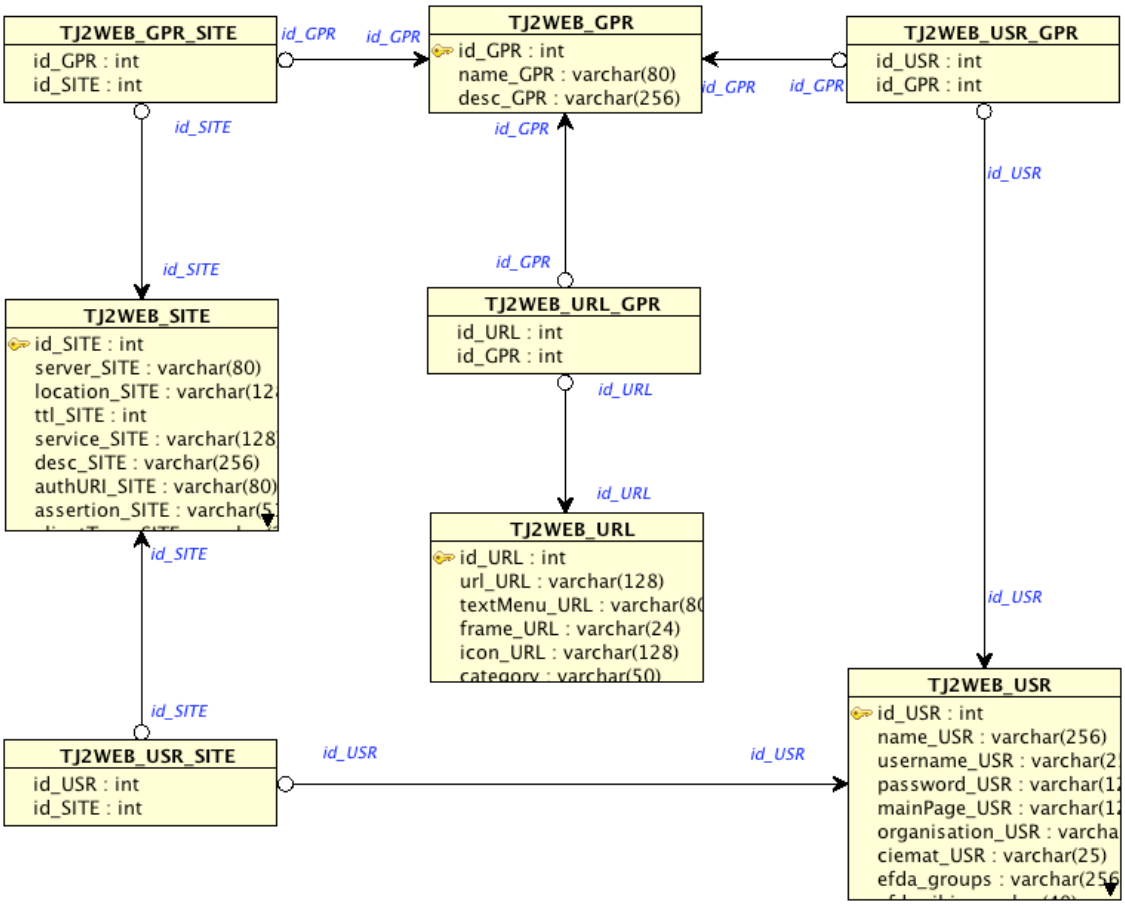

Fig. 3-10 *Diagrama entidad relación del repositorio de usuarios implementado para la integración de PAPI en el entorno de participación remota del TJ-II*

3.5. Agrupación de puntos de acceso (GPoA)

3.5.1. La funcionalidad GPoA

Una de los problemas más evidentes del sistema PAPI a nivel funcional, y que no supone en ningún caso un problema a nivel de seguridad, parte de su esquema de autorización basado en secuencias de tipo PUSH. En dicho esquema (como ya se ha comentado previamente) el usuario carga una serie de "tokens" firmados en la fase de autenticación, y cuando el usuario realiza una consulta solicitando un servicio, se anexa a la consulta el “token” correspondiente que permita la autorización de dicha petición. En el caso del sistema PAPI, los “tokens” firmados se corresponden a “cookies” cifradas que son anexadas en sucesivas consultas, y que son precargadas en la aplicación cliente del usuario en la fase de autenticación. El hecho de que se utilicen

"cookies" hace que sea necesario una "cookie" por cada zona web o servicio a proteger, es decir por cada PoA, ya que, como ya se ha mencionado previamente, sólo el recurso que genera una "cookie" puede hacer uso de ella. Por consiguiente, el número de "cookies" que pueden llegar a precargarse es excesivamente elevado y esto puede generar problemas de escalado.

Otro de los problemas derivados del diseño distribuido de PAPI radica en que cada recurso o servicio protegido (por cada PoA) tenga que ser registrado en cada uno de los servidores de autenticación válidos. Es decir, si un administrador instala un nuevo PoA para proteger un recurso, éste debe ser registrado en uno o varios de los servidores de autenticación válidos, ya que dichos servidores tendrán que generar cookies válidas para dicho PoA para aquellos usuarios que necesiten acceder a él. Esto genera un serio problema de gestión, ya que aunque se tenga un servidor de autenticación para una organización, no resulta flexible un modelo en el que se tenga que estar registrado en dicho servidor cada recurso que vaya a ser dado de alta. Por otro lado, una solución basada en un servidor central de registro, eliminaría el carácter distribuido de PAPI y lo inhabilitaría como solución de seguridad pra el entorno de participación remota del TJ-II.

Como solución al problema previamente planteado, se buscó un esquema que permitiera concentrar las cookies a precargar. Con este fin se dotó a los puntos de acceso de la capacidad de agruparse, de tal forma que con precargar la "cookie" correspondiente a un grupo de PoAs, el usuario se identificara frente a todos los PoAs del grupo. La implementación de esa idea se plasmó en el GPoA (Group Point of Access) [3] y que consiste en un nuevo componente en el sistema PAPI que agrupa a PoAs "hijos" bajo un GPoA "padre".

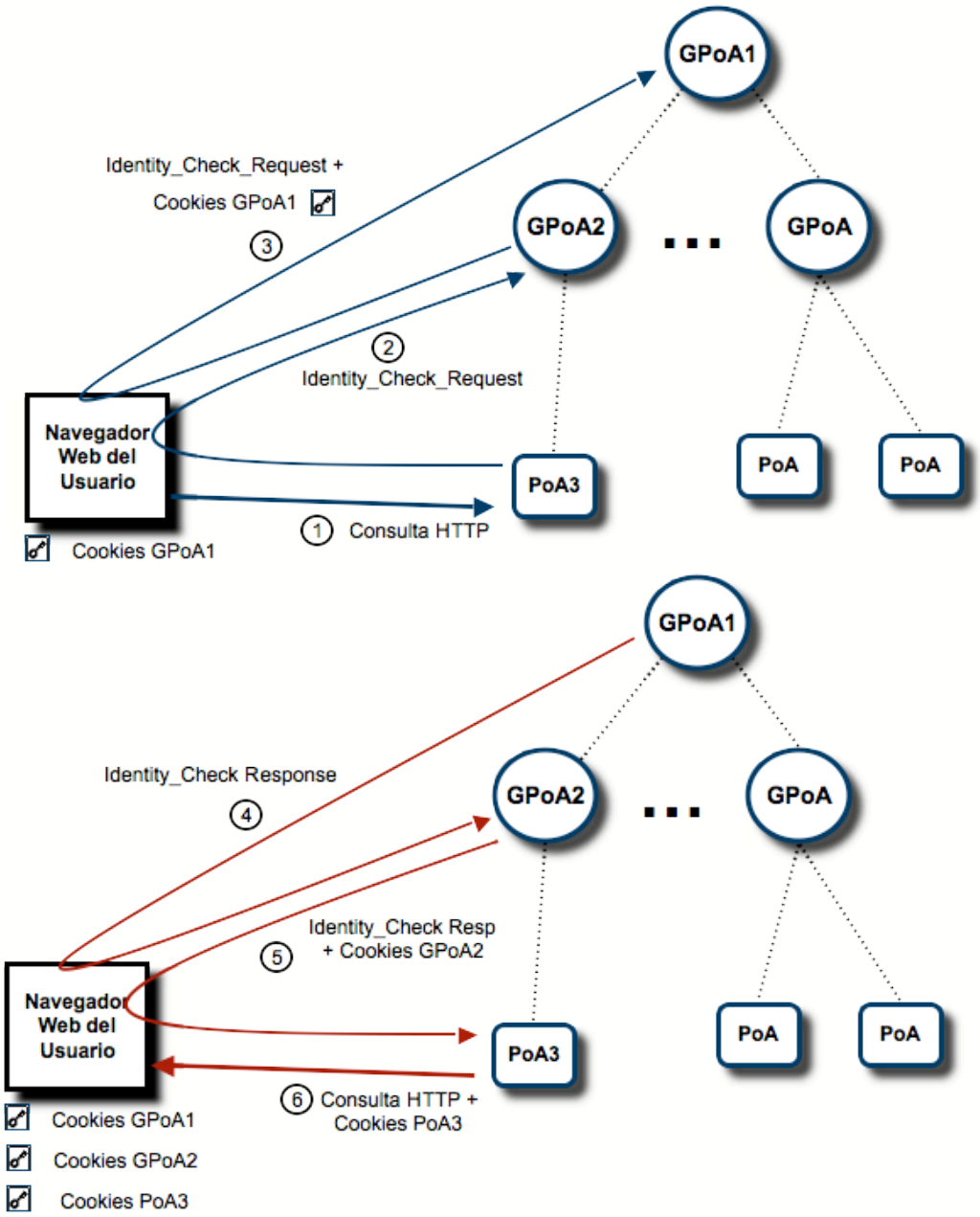

Fig. 3-11 Diagrama de colaboración que muestra el esquema de funcionamiento del GPoA basado en redirecciones HTTP**.**

Una consecuencia muy importante de este modelo es la posibilidad de agrupación en cascada como muestra el diagrama de colaboración de la Fig. 3-11. Esta capacidad permite que los PoAs se puedan agrupar y a su vez que grupos de PoAs puedan agruparse en un grupo de mayor nivel, y así sucesivamente sin un límite de agrupaciones. La única limitación proviene de la limitación en cuanto al número de redirecciones HTTP que es capaz de soportar la aplicación cliente del usuario.

3.5.2. El esquema de funcionamiento del GPoA

El diagrama de colaboración de la Fig. 3-11 muestra el funcionamiento de un PoA que está asociado a un GPoA padre, que a su vez esta asociado a otro GPoA de mayor nivel. Para explicar este diagrama se parte de un usuario que ya ha sido autenticado y ha precargado la "cookie" del GPoA raíz y el funcionamiento es el siguiente:

1. El usuario intenta acceder a un recurso o servicio y realiza una petición HTTP.
2. El PoA del recurso procesa la petición. Si todo es correcto deja continuar la petición. En caso contrario, si detecta que la petición no lleva asociada ninguna "cookie" PAPI ("Hcook") y si tiene configurado un GPoA padre, antes de dar la consulta como rechazada, redirecciona una petición de chequeo a dicho GPoA.
3. El GPoA recibe la petición de chequeo por parte del PoA hijo a través de la redirección del usuario. Si el usuario es identificado por el GPoA, es decir, recibió "cookies" de éste en algún momento y son correctos, el GPoA podrá responder al PoA o GPoA que le envió la consulta con el resultado de la identificación. En caso de que el GPoA no pueda identificar al usuario, pueden ocurrir dos cosas: si el GPoA tiene configurado un GPoA padre, volverá a redireccionar la petición de chequeo a éste, en caso contrario, devolverá un mensaje de error de identificación al PoA o GPoA hijo que le envió la consulta.
4. El PoA, o GPoA en su caso, recibe la respuesta del GPoA al que escaló la consulta de chequeo del usuario.
5. Si la respuesta corresponde a una identificación positiva:
6. Se generarán nuevas "cookies" que serán enviados en la respuesta al usuario.
7. Si se trata de un PoA final, éste permitirá continuar al usuario con la petición inicial.
8. Si la respuesta a la identificación es positiva:
    a. Si se trata de un PoA final, éste acabará de resolver la consulta original y devolverá el resultado al usuario.
    b. Si se trata de un GPoA, este redireccionará el resultado del chequeo al PoA (o GPoA) que envió la consulta.
9. Si la respuesta correspondiente a la identificación es negativa:

a. Si se trata de un PoA final, éste enviará una respuesta de petición no autorizada al usuario.

b. Si se trata de un GPoA, este propagará el resultado erróneo del chequeo al PoA (o GPoA) que envió la consulta.

3.5.3. El protocolo de comunicación con el GPoA

Existen básicamente dos tipos de mensajes en un esquema de agrupación de PoAs: el mensaje "Identity_Check_Request" (petición de chequeo de identidad) y el mensaje "Identity_Check_Response" (resultado del chequeo de identidad). El primero de ellos sirve para redireccionar la identificación de un usuario y se utiliza tanto entre un PoA y su GPoA como entre un GPoA y su GPoA padre. El segundo es consecuencia de una previa petición de identificación y sirve para comunicar el resultado de la identificación y se utiliza igualmente entre PoA y GPoA como entre GPoA y GPoA. Además en los casos de identificación positiva, este mensaje es utilizado para enviar nuevas "cookies" PAPI al usuario. Estas “cookies” serán anexadas en futuras consultas a este GPoA, permitirán la identificación del usuario a este nivel, evitando el redireccionamiento a otros GPoAs superiores.

Los campos del mensaje de consulta se muestran a continuación:

- ACTION: Acción solicitada. Para este tipo de consultas este campo toma el valor "CHECK".

- DATA: Identificador aleatorio que permite recuperar el estado de la consulta original que envió el usuario y así poder continuar con ella después de todas las redirecciones. El estado de la consulta original se almacena, antes de proceder a la redirección, en un sistema de almacenamiento local y este campo corresponde al valor de la clave para poder recuperar la consulta cuando así se requiera.

- URL: Este parámetro hace referencia a la URL justo anterior a esta petición de identificación, y es utilizada por el GPoA como dirección URL a la que enviar la respuesta.

El mensaje de respuesta consiste en una respuesta HTTP 302 de redirección a la URL que se especifica en la cabecera "Location". Los campos del mensaje de respuesta son:

- Location: Campo tipo cabecera HTTP donde se representa la dirección URL a la que enviar la respuesta y que debe corresponder al campo URL de la consulta correspondiente.
- Set-Cookie: Campo opcional de tipo cabecera HTTP que se envía si la identificación del usuario ha sido correcta. Contiene nuevos valores para las cookies PAPI asociadas al GPoA que genera la respuesta y que serán anexados en la próxima consulta a este GPoA.
- ACTION: Acción del mensaje (Para estas consultas este campo contiene el valor "CHECKED")
- DATA: Este campo contiene la codificación en "Base64" del resultado de la petición de identificación del usuario, cifrado mediante RSA, que se describe en detalle en el punto 3.5.5. El resultado de la petición de identificación, que es enviado dentro de este campo, se compone de una cadena texto con una serie de datos:
    - "ERROR + NULL + Fecha de creación de este mensaje + El campo DATA de la consulta correspondiente a esta respuesta: en caso de que la identificación haya sido negativa.
    - Identificador de usuario + Fecha de expiración de la identificación + Fecha de creación de este mensaje + El campo DATA de la consulta correspondiente a esta respuesta": en caso de identificación positiva.

3.5.4. Sistema de redirección de PAPI

El esquema de funcionamiento del GPoA se basa en la capacidad de redirección que soporte la aplicación cliente y el protocolo utilizado. En este caso, HTTP da la posibilidad de utilizar redirecciones dentro de su estándar. Para la implementación del esquema GPoA, se desarrolla toda la tecnología de redirección para consultas de tipo GET y POST, que permite, después de una identificación positiva por parte del GPoA padre, bien propagar el resultado de la identificación a PoAs hijos, bien continuar con la consulta original que realizó el usuario. Este tipo de funcionamiento obliga a mantener el estado de la consulta hasta recibir una respuesta del GPoA padre. En la Fig. 3-12 se muestra el diagrama de actividad correspondiente a la nueva funcionalidad, que aporta a PAPI una gran flexibilidad de organizar y estructurar los PoAs de una organización. En

el diagrama de actividad original *Fig. 2-4*, este proceso enlazaría en el punto donde se detecta que la "Hcook" no es correcta. Es decir, en vez de generar una respuesta HTTP prohibiendo el acceso, se redirige al usuario al GPoA padre.

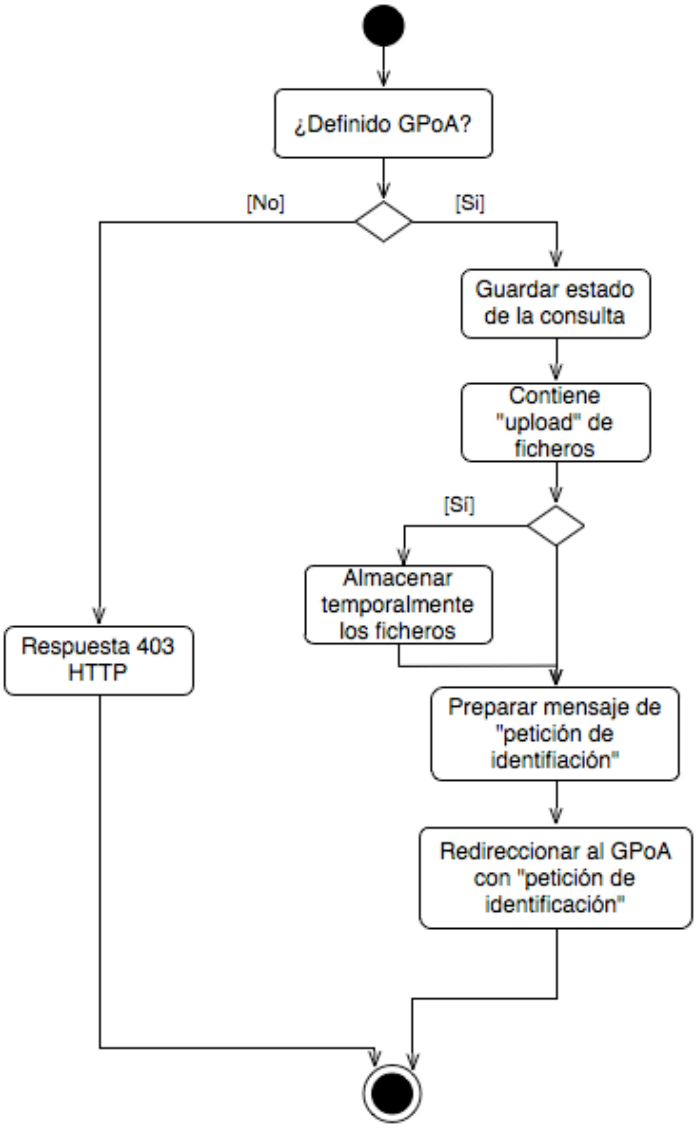

Fig. 3-12 *Diagrama de actividad correspondiente al proceso de redirección de un usuario a un GPoA para comprobar su identidad*

Para guardar el estado de las consultas, se ha utilizado un sistema de base de datos local embebido, que soporta concurrencia tanto para lectura como para escritura. Asimismo, y con el objetivo de utilizar un sistema gestor de base de datos lo más reducido posible, se ha evitado la utilización de SQL y se ha optado por un sistema de acceso a datos asociativo, es decir, escrituras tipo set("clave","valor") y lecturas de tipo "valor" = get("clave"). Como implementación se ha integrado un sistema de base de

datos Berkeley con soporte de concurrencia y con interfaz para Perl (lenguaje de implementación del PAPI).

La clave para almacenar el estado de la consulta es generado aleatoriamente con GUID (Global Unique Identifiers), que garantiza identificadores únicos hasta el año 3400 D.C. y su valor es incluido en el campo DATA de redirección al GPoA padre. La información de la consulta que se almacena antes de la redirección se detalla a continuación.

Para consultas tipo "GET" y "POST":

- Headers: Vector compuesto por todas las cabeceras HTTP de la consulta con sus valores correspondientes.

- Method: Método de la consulta "GET" o "POST"

- URI: URL origen de la consulta (sin la parte de argumentos)

- Args: Vector compuesto por los parámetros incluidos dentro de la URI origen de la consulta y sus valores correspondientes. Parámetros típicos de consultas tipo GET pero que también pueden aparecer en consultas tipo POST.

- Filename: Nombre del fichero local asociado a la URL de la consulta.

Exclusivamente para consultas tipo "POST":

- Post_params: Vector de parámetros (y sus correspondientes valores) asociados a la consulta pero que son recibidos utilizando el mecanismo de paso de parámetros "POST".

En el caso de consultas que incluyen el envío de uno o más ficheros, una solución derivada del sistema de base de datos Berkeley no resultaba viable. Para este tipo de consultas en las que se realizan cargas (subidas) de ficheros, se ha implementado un buffer temporal que permite mantener copia de dichos ficheros incluidos en la consulta hasta que ésta es resuelta.

Para las consultas tipo GET el mecanismo de implementación de módulos en Apache permite recargar el estado de la consulta originaria que envió el usuario y continuar con ella sin que el usuario perciba todo lo que ha ocurrido entre medias. Para conseguir ese mismo efecto con las consultas tipo POST, no basta con restablecer el estado de la consulta originaria, sino que además hay que utilizar un cliente HTTP que

construya una nueva consulta a partir de la información almacenada. De esta forma es posible pasar como POST todos los parámetros almacenados de la consulta originaria y sus "subidas" de ficheros.

3.5.5. Cifrado de clave pública en los GPoAs

El esquema de agrupamiento de PoAs requiere de un esquema de seguridad que permita asegurar la comunicación entre un GPoA y los PoAs (o GPoAs) hijos. Analizado desde otro punto de vista, es un caso similar al que ocurre en la comunicación entre un servidor de autenticación y los PoAs a los que solicita la generación de nuevos tokens de sesión (cookies PAPI) cuando un usuario ha sido correctamente autenticado. En el caso del GPoA también se produce una autenticación del usuario por parte del GPoA, sólo que en este caso resulta transparente, y como credenciales, el usuario entrega cookies PAPI previamente cargadas. Se puede decir que desde el punto de vista del servidor de autenticación, el GPoA se comporta como un autenticador "delegado". Como consecuencia de la respuesta del GPoA al PoA hijo, este va a entregar “cookies” válidas al usuario. Por lo tanto, la securización de dicha respuesta resulta crucial, ya que si no, cualquiera podría generar un mensaje de GPoA falso que le daría acceso a los recursos de PoAs hijos.

Al igual que ocurre en el esquema "Servidor de Autenticación - Punto de Acceso", los mensajes enviados desde el GPoA a su PoA hijo deben poder ser autenticados y se debe poder garantizar su integridad, es decir, se debe poder comprobar si el que lo envía es realmente el GPoA, y además, se debe poder comprobar si el mensaje no ha sido modificado. Como solución se puede optar por un esquema de cifrado con clave simétrica en el que PoA y GPoA comparten una clave común, o un esquema de cifrado basado en clave asimétrica en la que el GPoA, emisor del mensaje, posee la clave privada y el PoA, receptor del mensaje, posee la clave pública. Como ya se ha comentado previamente, en el caso de cifrado simétrico la clave con la que se cifra el mensaje debe ser la misma con la que se descifra, mientras que en clave asimétrica existe un par de claves, y si se cifra con una de ellas, se debe descifra con la otra.

En este caso existe un cifrador (GPoA) y varios descifradores (PoA). Si se optara por un esquema de cifrado simétrico habría que generar un par de claves por cada nuevo PoA hijo dentro del grupo, y cada vez que se instalara un PoA (para proteger por ejemplo una nueva zona web), habría que declararlo en el GPoA padre y generar nuevas

claves, lo que volvería la solución poco flexible y compleja de gestionar. Por el contrario si se opta por clave asimétrica, el GPoA cifraría todos los mensajes con su clave privada (con lo que se garantizaría la autenticidad e integridad del mensaje), y el PoA receptor utiliza la clave pública del GPoA para descifrar el mensaje. Una consecuencia de este modelo es que sólo es necesario un par de claves para todos los PoAs y si se desea incorporar un nuevo PoA, no habría que declararlo, ya que con que el PoA instalara la clave pública del GPoA padre es suficiente. Cara a la concepción del sistema PAPI, agrupamiento basado en GPoA, potencia de forma fundamental el carácter distribuido de la solución ya que disminuye notablemente la dependencia y relación existente entre los PoAs y el servidor de autenticación, permitiendo instalar gran cantidad de PoAs sin coste administrativo para el mantenimiento del servidor de autenticación y el repositorio de usuarios de la organización. En la sección correspondiente a la implementación de la federación EFDA con PAPI, se extiende el concepto GPoA y se puede apreciar todo su potencial a la hora de desarrollar soluciones distribuidas.

De los posibles esquemas de cifrado con clave asimétrica se opto por RSA, ya que es el mismo que se utiliza para la comunicación SA -> PoA. Para su implementación se utiliza la librería OpenSSL junto con un adaptador en Perl para utilizar librerías de C. Como sistema de representación del mensaje resultante en ASCII (para poder ser enviado en un mensaje HTTP) se optó por una codificación "Base64".

3.6. Nuevo gestor de control de acceso

A la hora de establecer políticas de control de acceso a recursos o servicios web, se requiere en muchas ocasiones definir reglas de control de acceso que dependan bien de parámetros relacionados con información relativa al usuario (organización a la que pertenece, cargo que ocupa, proyectos en los que trabaja, etc.), bien de parámetros relacionados con información incluida en la consulta (tipo de señal que se quiere ver, instrumental al que se le envía la consulta, acción a realizar, etc.), o incluso de parámetros relacionados con las condiciones de la propia consulta (IP desde donde se realiza la consulta u hora a la que se realiza). Con el objetivo de dotar a los PoAs de capacidad para implantar las políticas complejas mediante las reglas antes descritas, se ha desarrollado un nuevo subsistema gestor de control de acceso para los PoAs.

Las reglas de control de acceso se definen, dentro del fichero de configuración, mediante etiquetas "Access_Rule" en cada una de las secciones a las que afecta. En concreto su sintaxis es:

<Access_Rule action="(accept|reject)" > Access Rule </Access_Rule>

Mediante la propiedad "action" se define la consecuencia de que la regla de acceso se cumpla. Si toma el valor "accept" se fuerza a que la consulta se acepte, mientras que "reject" fuerza al PoA a rechazar la petición. Para poder establecer un orden lógico de procesamiento de reglas de acceso, se ha establecido que sea por orden de aparición en el fichero de configuración de PAPI.

Un aspecto importante del nuevo sistema de acceso es que una regla de acceso es evaluada en cada consulta sólo si contiene el parámetro "%_URL" o algún parámetro de tipo "%req_..". En caso contrario, la regla de acceso sólo es evaluada en el momento de generar o rotar las “cookies” de un usuario. Este esquema de funcionamiento libera de forma significativa la carga del PoA en su tarea de control de acceso.

3.6.1. Sintaxis de las reglas de control de acceso

La gramática aplicable a las reglas de acceso descritas mediante la sintaxis EBNF (Extended Backus–Naur Form) [39], comúnmente utilizada para definición de gramáticas, queda de la siguiente forma:

```
exp =       '[' , exp , ']'
       | exp log exp
       | 'NOT', exp
       | elem cond elem
       | SFUNC;

log    =    'AND' | 'OR' ;

cond   =    '-eq' | '-lt' | '-gt'
            | '-le' | '-ge' | '-regex'
```

```
            | '=' | '-in' ;

elem   =    NUM | STRING | par ;

par    =    '%' , VAR | '%req_'VAR | '%_NOW_mday'
       | '%_NOW_mon' | '%_NOW_year' | '%_NOW_wday'
       | '%_URL' | '%_AS';
```

Como se puede ver en la gramática definida, existen 4 grupos principales:

- **Operadores booleanos**: 'AND', 'OR' y 'NOT.
- **Operadores condicionales numéricos**: '-eq' (igual que), '-lt' (menor que), '-gt' (mayor que), '-le' (menor o igual que), y '-ge' (mayor o igual que).
- **Operadores condicionales para cadenas de caracteres**:
    - 'regex': Realiza un mapeo del término izquierdo sobre la expresión regular definida en el término derecho.
    - '=': Compara dos cadenas de caracteres.
    - '-in': Comprueba si el término izquierdo de la expresión se encuentra como elemento en la lista definida por el término derecho.
- **Funciones Especiales SFUNC que representa las funciones**:
    - IPmatch(<rango de IPs>): devuelve "true" en caso de que la IP origen de la consulta se encuentre dentro del rango de IPs que se le pasa como argumento a la función. Devuelve false en caso contrario.
    - InDates(<Date1>,<Date2>): devuelve "true" en caso de que la fecha actual de la consulta se encuentre dentro del rango de fechas definido por los argumentos de la función "Date1" y "Date2". Devuelve "false" en caso contrario.

Además existen parámetros con una sintaxis predefinida que pueden ser utilizados en la descripción de las reglas:

- **Parámetros predefinidos de la consulta**:
  - Sobre la fecha de la consulta: %_NOW_mday', '%_NOW_mon', '%_NOW_year', '%_NOW_wday'
  - La URL de la consulta: '%_URL'
  - El servidor de autenticación del usuario: '%_AS'
- **Parámetros de usuario: Son variables del tipo "%VARIABLE"** y se corresponden con parámetros, con el mismo nombre, dentro de la información que PAPI da sobre el usuario. Esta información no varía entre consultas, pero sí puede variar entre autenticaciones.
- **Parámetros de consulta: Son variables del tipo: "%req_VARIABLE"** y se corresponden con parámetros que son pasados (bien como GET o POST) en la consulta. Esta información puede variar entre consultas.

3.6.2. Implementación

Para la implementación del motor de autorización se ha utilizado una versión modificada del módulo Perl "Parser-BooleanLogic" que permite construir una estructura de árbol binario a partir de una expresión que contiene "operandos", "operadores", funciones lógicas ("AND" y "OR"), y paréntesis. Para ello, implementa un analizador sintáctico basado en la construcción y recorrido de un árbol binario en modo "post-orden", es decir, primero se recorre en profundidad cada rama de un nodo y finalmente se procesa la raíz del nodo. Las dos funciones principales de BooleanLogic que han sido utilizadas para la implementación del motor de autorización han sido:

- "as_array": Esta función genera a partir de una cadena de caracteres que se le pasa como entrada, una estructura de árbol binario en la que los nodos corresponden con los tokens AND y OR dentro de la cadena de estrada y el resto de símbolos (salvo los paréntesis) se consideran nodos finales. En esta función se realizó una de las modificaciones al código original. En concreto se incluyo el operador 'NOT' y se posibilitó el uso de los corchetes '[' , ']' como elementos delimitadores de paréntesis, ya que las funciones especiales (antes comentadas) IPrange() y InDate() utilizan los paréntesis en su sintaxis.

- "solve": Esta función resuelve las expresiones booleanas del árbol binario realizando un recorrido "post-orden". Es decir, para un nodo, primero resuelve el valor booleano de la rama izquierda, después resuelve el valor booleano de la rama derecha, y finalmente devuelve el valor resultante de aplicar la operación lógica de la raíz a ambas ramas. En caso de ser nodo final, "solve" utiliza una función "callback" que aplica a la expresión almacenada en dicho nodo. Otro de las funciones claves en la implementación del motor de autorización es la función "callback" que, como se ha comentado previamente, es utilizada por la función "solve" para calcular el valor booleano de una expresión almacenada en un nodo final del árbol binario a procesar. Los pasos para el cálculo de un nodo final son los siguientes:

1- Si hay un operador binario ("-eq", "-gt", "in" , etc.), primero se evalúan las expresiones de ambos lados del operador y se devuelve el resultado del operador con dichos valores.

2- Si no hay un operador binario, se sustituyen variables en la expresión de tipo "%..." por su correspondiente valor.

3- Si es una función especial ("IPrange" o "InDate") se devuelve el resultado de dicha función.

4- En cualquier otro caso se devuelve el valor literal de la expresión, sea número o cadena de caracteres.

3.7. Integración de aplicaciones JAVA

Las aplicaciones JAVA desarrolladas en el entorno de participación remota del TJ-II se caracterizan por seguir el modelo cliente - servidor, en el que la aplicación cliente basa su funcionamiento en la realización de consultas a uno o varios servidores remotos, bien para obtener datos, bien para ordenar ciertas acciones sobre las bases de datos o sistemas del TJ-II. En el caso de estas aplicaciones del entorno de participación remota, las conexiones con los servidores las realizan utilizando HTTP o HTTPS [40] como protocolo base de las comunicaciones, con el objetivo de estandarizar el protocolo de acceso a los servicios del TJ-II y facilitar el acceso desde localizaciones remotas con cortafuegos de por medio.

Una de las estrategias para la integración de las aplicaciones JAVA en PAPI es la adaptación de librerías que puedan ser utilizadas por aquellas como librerías base para sus conexiones HTTP, sustituyendo a la clase "java.net.HttpURLConnection" existente en las librerías primitivas de la máquina virtual de JAVA. Con ello se garantiza, por un lado la compatibilidad de la aplicación con los requisitos funcionales de PAPI como manejo de cookies y mecanismos de redirección, y por otro lado, que todas las aplicaciones manejan el mismo repositorio de cookies. Esta última condición resulta fundamental tanto para el sistema de sesión asociado al usuario, como para el sistema de "Single-Sign-On" de PAPI [41].

Al comienzo del proyecto, la librería estándar para conexiones HTTP de JAVA no manejaba cookies, por lo que se optó por comenzar adaptando a PAPI, en una primera fase, otras librerías como son HTTPClient de Ronald Tschalär y la librería HttpClient del proyecto Jakarta. Esta última es la librería base para importantes desarrollos del proyecto Jakarta, como son: Apache, Tomcat, o el protocolo de llamada a procedimiento remoto: XML-RPC [42]. A partir de la versión 1.5 de JAVA, su librería estándar comenzó a manejar cookies en sus conexiones HTTP, por lo que en una segunda fase, esta fue adaptada a PAPI.

Como característica de este esquema de control de acceso, conviene destacar que es un esquema de seguridad orientado a conexión y no orientado a aplicación. Es decir, que en este caso no se protege la descarga de la aplicación (que puede utilizada con posterioridad por cualquiera), ni se protege el lanzamiento de la aplicación (que puede dejarse abierta después de finalizar la sesión), sino que lo que se protege es cada una de las conexiones que realiza la aplicación. Como consecuencia de ello, si un usuario sale de su sesión, la aplicación queda inutilizada en lo que a conexiones al servidor se refiere. De la misma forma, si el usuario vuelve a autenticarse, la aplicación vuelve a estar completamente operativa.

3.7.1. El repositorio de cookies PAPI

La estrategia que se ha seguido, tal y como muestra la Fig. 3-13, para integrar las librerías antes mencionadas en el mecanismo de “Single-Sign-On” de PAPI ha sido la de proveer a todas las aplicaciones de un repositorio común de credenciales. Para ello se decidió implementar dicho repositorio, y proveer a cada librería de las clases y mecanismos necesarios para dotar a los métodos que requieran el manejo de las cookies

de un acceso transparente y concurrente al repositorio común de tokens de sesión, implementados en este caso como cookies cifradas.

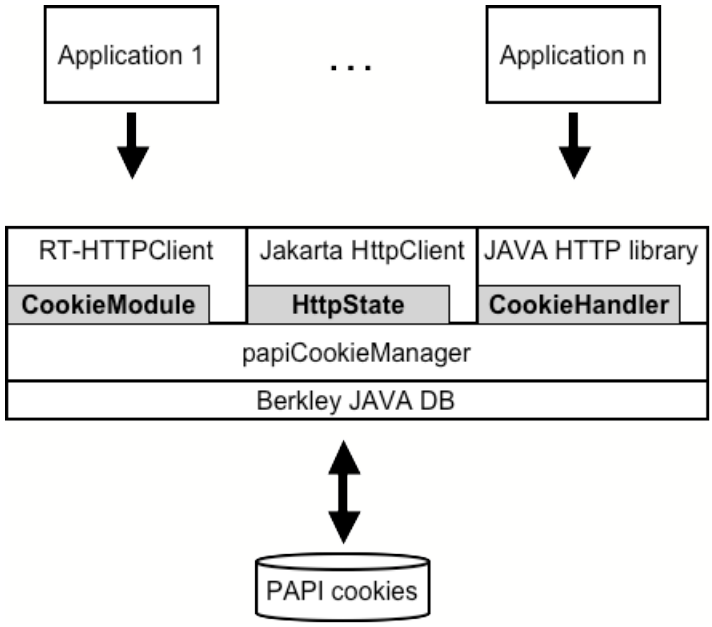

Fig. 3-13 *Arquitectura de la solución para la integración de las aplicaciones basadas en conexiones HTTP en PAPI*

Los requisitos que debe cumplir el repositorio de "cookies" PAPI son:

- **Unicidad y accesibilidad**: Cada usuario debe tener su propio repositorio, y este debe ser accesible por cualquier aplicación o proceso propiedad de dicho usuario.

- **Concurrencia a nivel de threads y procesos**: Este repositorio debe ser accesible de forma concurrente por diferentes aplicaciones, así como por diferentes threads dentro de una misma aplicación.

- **Independencia de la plataforma**: El repositorio debe ser lo más independiente de la plataforma a instalar y compatible con los sistemas operativos Windows, Linux y Mac OSX.

- **Transparencia**: Tanto el repositorio como cualquier sistema que lo gestione deben ser lo más transparente para el usuario como sea posible y con un proceso de instalación que requiera mínima interacción con el usuario.

En base a estos requisitos se opta por realizar la implementación del repositorio de cookies utilizando, como tecnología soporte, el sistema de base de datos Berkeley DB. Este sistema permite la implementación de bases de datos embebidas con un esquema

de acceso a datos basado en pares clave/valor, lo que simplifica enormemente el motor del sistema gestor de base de datos al evitar lenguajes complejos de consulta como SQL. Dentro de las diferentes implementaciones del sistema Berkeley DB, para el este desarrollo se optó por la implementación realizada en JAVA, lo que le da independencia frente a la plataforma donde va a ser utilizado.

Además, la sencillez de su sistema gestor, y su concepción como base de datos embebida, permite que si la instalación se realiza desde una aplicación, ésta resulte muy sencilla y transparente para el usuario, reduciéndose a un conjunto de ficheros con localización en el espacio de ficheros del usuario. Finalmente, cabe destacar que el sistema de base de datos elegido es transaccional por lo que se garantiza el acceso concurrente, tanto a nivel de múltiples hilos de ejecución, como a nivel de múltiples aplicaciones.

3.7.1.1. Interfaz de acceso común para el repositorio de cookies

Con el fin de proveer de una interfaz de acceso común al repositorio de cookies PAPI a todas las librerías y aplicaciones que así lo requirieran, se desarrolló una librería de acceso en JAVA llamada "papiCookieManager". Además de ser un punto de acceso común, permite también abstraer la funcionalidad del repositorio de cookies del sistema seleccionado para su implementación (en este caso JAVA Berckley DB).

La Fig. 3-14 muestra el diagrama de clases de la librería "papiCookieManager". Tal y como muestra el diagrama, hay dos clases "PapiCookie" y "CookiesDB" que son las que soportan todo el trabajo de interfaz, y usan, principalmente, el paquete "com.sleepycat" perteneciente a la API de JAVA Berkeley DB. Aparte de estas clases, hay dos clases "CookieBinding" y "CookieKeyBinding" que realizan un trabajo de "serialización" de la clave y los datos a almacenar en la Berkeley DB. Para ello, extienden la clase "TupleBinding" del paquete "com.sleepycat".

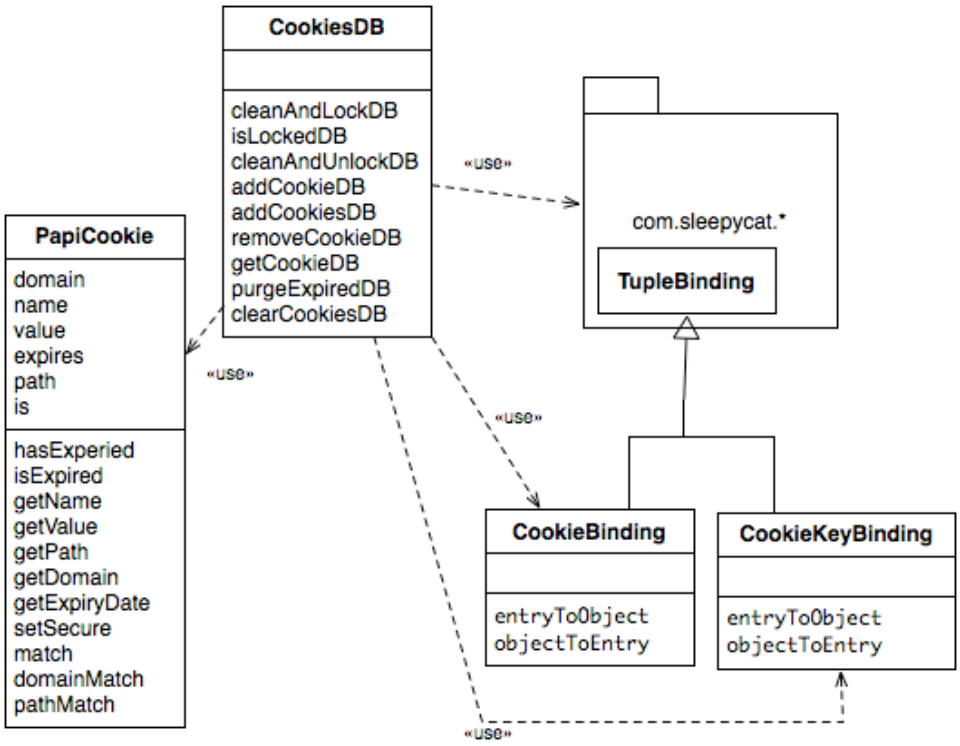

Fig. 3-14 *Diagrama de clases correspondiente a la sección de la librería papiCookieManager realacionado con el repositorio único de cookies.*

Los objetos de la clase "**PapiCookie"** representan cookies PAPI, que son los objetos base manejados en la librería "papiCookieManager". Para adaptar una aplicación o librería a este interfaz tendría, bien que manejar este tipo de objetos, bien realizar una transformación o traducción de "PapiCookies" a los objetos "cookie" que maneje. Los atributos de un objeto "PapiCookie" se corresponden con los de una típica "cookie" HTTP:

- **domain**: Dominio o nombre del servidor sobre el que esta "cookie" tiene competencia.

- **name**: Nombre de la "cookie". En el caso de cookies PAPI, serán del tipo "L_Hcook_..." o "L_Lcook_...".

- **value**: Valor de la "cookie".

- **expires**: Fecha y hora de expiración de la "cookie". Todas las fechas manejadas por PAPI se expresan en GMT (Greenwich Mean Time) para mantener la compatibilidad entre servidores de diferentes zonas horarias.

- **path**: Este campo representa la zona afectada por la "cookie". Todas aquellas consultas cuya URL tenga un path que empiece como el parámetro "path" de la "cookie", deberán incluir dicha "cookie" en la consulta HTTP.

- **is**: Este campo determina si la "cookie" es aplicable sólo a consultas HTTPS.

Respecto a los métodos de la clase "PapiCookie", la mayoría se corresponden con métodos tipo "get...." utilizados para obtener el valor de los atributos del objeto. Aparte de éstos, hay un método "hasExpired", que comprueba si la "cookie" sigue siendo válida en función de su campo "expires" y la hora y fecha en el momento de la comprobación, y un método "match" utilizado para comprobar si la "cookie" debe incluirse en una consulta con cierta URL.

La clase **CookiesDB** encapsula un conjunto de métodos necesarios para trabajar con el repositorio de cookies y cuyas órdenes son traducidas al sistema repositorio subyacente (en este caso Berkeley DB). Existe un primer conjunto de métodos destinados a almacenar, borrar y recuperar cookies del repositorio: addCookieDB, addCookiesDB, removeCookieDB, getCookieDB. Hay un segundo grupo destinado a mantener el repositorio lo más actualizado posible: clearCookiesDB y purgeExpiredDB. Y para terminar, hay un tercer grupo destinado a inicializar, vaciar y mantener la coherencia del repositorio de cookies en sucesivas consultas HTTP. El método "cleanAndLock" elimina las "cookies" PAPI del repositorio y fija un valor que bloquea la base frente a nuevas "cookies" PAPI en sucesivas consultas, aunque requieran ser escritas. Se corresponde con un proceso de "logout" de PAPI. El método "cleanAndUnlock", elimina las cookies PAPI del repositorio y abre la base a nuevas cookies PAPI (se corresponde un proceso de autenticación exitoso). Finalmente, el método "isLockedDB" informa sobre si la base de cookies está bloqueada y por tanto no pueden ser escritas nuevas cookies.

La clase **CookieBinding** y **CookieKeyBinding** se utilizan para "serializar" (codificar) el objeto PapiCookie y la cadena de caracteres que forman la clave del par (clave, PapiCookie) a almacenar en la JAVA Berkeley DB, con la que se implementa el repositorio de cookies. En realidad se trata de codificar el objeto a almacenar y la clave con la que se almacena, a objetos tipo "TuplaInput" y desde objetos tipo "TuplaOutput", para las operaciones de escritura y lectura respectivamente. Para ello se utilizan operaciones típicas de escritura y lectura en stream de JAVA.

3.7.2. La librería RT-HTTPClient

La librería RT-HTTPClient de Ronald Tschalär [28] fue la primera elegida para su adaptación a PAPI. En el ámbito de las librerías de uso gratuito y código abierto, las razones para elegir esta librería HTTP en JAVA fueron:

- **Protocolo HTTP en versiones 1.0 y 1.1**: La librería soporta las dos versiones más extendidas del protocolo.

- **Manejo de cookies**: Fue una de las primeras librerías HTTP en JAVA con manejo de cookies HTTP.

- **Mecanismos de redirección**. Esta librería soporta los mecanismos de redirección propios del protocolo HTTP.

- **Soporte de conexiones HTTPS**. Una vez más, esta librería fue una de las primeras en el soporte de protocolo de conexión HTTPS, lo que resulta fundamental en un sistema concebido como infraestructura de seguridad.

- **Soporte de aplicaciones multithread**. La implementación de la librería soporta memoria compartida y código reentrante propios de aplicaciones "multi-thread".

Otra característica muy importante de esta librería es que puede ser utilizada en una aplicación que usa la clase estándar "HttpURLConnection" sin necesidad de recompilar ni modificar dicha aplicación. En concreto, basta con asignar a la variable de sistema "java.protocol.handler.pkgs" el valor "HTTPClient", para que la aplicación haga uso de la librería RT-HTTPClient en sus conexiones HTTP.

Cara a la integración de "papiCookeiManager" en la librería RT-HTTPClient, existe un aspecto que ha determinado todo el proceso. Se ha focalizado la integración en la parte del código con menor impacto en el resto del código de la librería, es decir, con menor grado de acoplamiento. Gracias al buen diseño de la librería, ésta tiene un alto grado de modularidad y por tanto facilita en gran medida su mantenibilidad y modificación. En concreto todos los casos de uso relacionados con el manejo de cookies hacen referencia en última instancia a las clases Cookie y CookieModule, por lo que la adaptación de la librería se centró en estas dos clases.

La primera clase implementa el objeto "Cookie" manejado por el resto de clases de la librería. En ella se incluyen, tanto los atributos del objeto (típicos de un objeto

"Cookie") junto con todos los métodos necesarios para su asignación y lectura. Como se puede ver en el diagrama de clases que muestra la

Fig. 3-15, tanto los atributos como métodos de la clase se corresponden prácticamente a los de la clase "PapiCookie" en la librería "papiCookieManager". Por mantener el código original lo más inalterado posible, se optó por dejar esta clase intacta y se trasladó los métodos de conversión de cookies, entre "Cookie" y "PapiCookie", necesarios para poder utilizar los métodos de "CookiesDB".

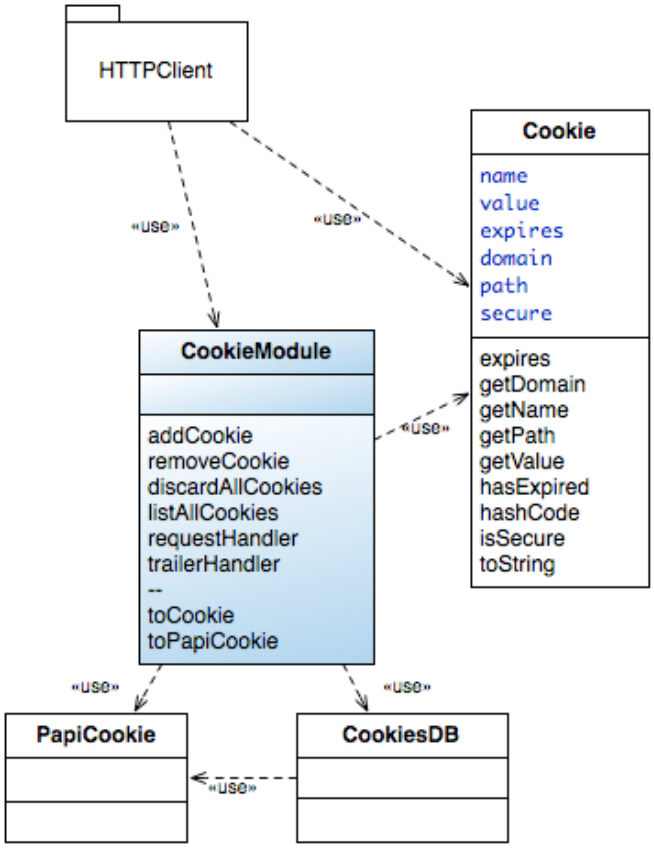

Fig. 3-15 *Diagrama de clases correspondiente a la parte de la librearía RT-HTTPClient relacionada con el manejo de cookies*

La segunda clase "CookieModule", que también se muestra en la

Fig. 3-15, implementa una serie de métodos para el manejo de cookies: "addCookie", "removeCookie", "discardAllCookies", "listAllCookies", suficientes para poder trabajar con un repositorio de cookies. Además incluye dos métodos que hacen de

enlace con el resto de la librería HTTPClient: "requestHandler" y "trailerHandler". Estos dos métodos son invocados por otras clases de la librería cuando hay que construir una consulta HTTP para ser enviada, o cuando se ha recibido una respuesta HTTP. De esta forma, todos los aspectos relacionados con el manejo de cookies se centran en estos dos métodos.

Como resultado del análisis de la librería y las dependencias entre sus clases y con respecto a las cookies, la adaptación de la librería se ha realizado modificando exclusivamente la clase "CookieModule". Para mantener el código original de HTTPClient lo más inalterado posible, se optó por dejar la clase "Cookie" intacta y se trasladó los métodos de conversión de cookies, entre "Cookie" y "PapiCookie", necesarios para poder utilizar los métodos de CookiesDB a la nueva clase "CookieModule". Como resultado, la nueva clase CookieModule mantiene los mismos métodos que tenía (cambiando parte de su código) y además incluye dos nuevos métodos de traducción de cookies: "toCookie" y "toPapiCookie".

Respecto al código modificado en los métodos antiguos, en las partes donde se realizaba lectura y escritura de cookies, sólo se han realizado modificaciones respecto a cómo se lee y se escriben las "cookies". El cambio más significativo proviene de que la librería HTTPClient original mantiene una estructura en memoria de las cookies almacenadas. En cambio, en la nueva versión se fuerza a que todas las cookies se almacenen en el repositorio central de cookies, sobre todo para mantener la coherencia entre las diferentes aplicaciones que hacen uso de él.

3.7.3. La librería Jakarta HttpClient

Esta librería forma parte del proyecto Jakarta y es una de las librerías HTTP referencia en JAVA. La principal razón que motivó en un principio la integración de esta librería en PAPI fue la necesidad de dotar a la librería de acceso a datos SDAS (Shared Data Access System) [43] desarrollada en el IST (Instituto Superior Técnico), de una infraestructura de seguridad compatible con el entorno del TJ-II. SDAS utiliza como protocolo base XML-RPC y más concretamente su implementación JAVA desarrollada dentro del proyecto Jakarta, que a su vez utiliza HttpClient como librería para sus conexiones HTTP.

Como ya se ha comentado, la librería jakarta HttpClient es una de las librerías de referencia en la implementación del protocolo HTTP en JAVA. Esto fue una motivación adicional a la hora de integrar esta librería en PAPI. Como principales funcionalidades tiene:

Implementa las versiones HTTP 1.0 y 1.1: soporta ambos protocolos.

Soporta el protocolo HTTPS: permite conexiones HTTP sobre SSL.

Soporta redireccionamiento: implementa los sistemas de redireccionamiento contemplados en el estándar HTTP.

Soporta conexión HTTP y HTTPS a través de proxies: es capaz de manejar conexiones a través de proxies, siendo capaz de utilizar "CONNECT" en el caso de conexiones HTTPS.

Soporta aplicaciones multithread: implementa código reentrante y sistema de bloqueos para este tipo de aplicaciones.

Soporta manejo de cookies: implementa mecanismos para el manejo de cookies, siendo capaz de interpretar cabeceras "Cookie" (en consultas) y "Set-Cookie" (en respuestas).

Multipart-form POST para envío de ficheros: es capaz de realizar envíos de ficheros a través de HTTP mediante el mecanismo POST en modo "multipart-form".

Respecto a la integración de "papiCookeiManager" en la librería "HttpClient", y al igual que ocurrió previamente en la integración de la librería RT-HTTPClient, ha prevalecido el criterio de la mantenibilidad. Para ello, las modificaciones se centran en las partes del código relacionadas con el manejo de cookies y con menor grado de acoplamiento respecto al resto del código, es decir, en el que las modificaciones tengan el menor impacto respecto al resto del código de la librería. Una vez más, quizá por ser JAVA un lenguaje basado en el paradigma de la programación orientada a objetos, y al igual que ocurrió con RT-HTTPClient, el código de la librería presenta un alto grado de modularidad y por tanto facilita en gran medida su mantenibilidad y modificación. En concreto todos los casos de uso relacionados con el almacenamiento y recuperación de cookies, hacen referencia a las clases "Cookie" y "HttpStatus", por lo que la adaptación de la librería se centró en estas dos clases.

La primera clase implementa el objeto Cookie manejado por el resto de clases de la librería. En ella se incluyen, tanto los atributos del objeto (típicos de un objeto Cookie) junto con todos los métodos necesarios para su asignación y lectura. Como se puede ver en el diagrama de clases que muestra la

Fig. 3-16, prácticamente todos atributos de la clase tienen su correspondiente en la clase PapiCookie. Respecto a los métodos, la clase Cookie tiene los típicos métodos "get", "set" e "is" para leer, asignar y comprobar el valor de los diferentes atributos. Un aspecto singular de la clase Cookie respecto a PapiCookie, es que los atributos "name" y "value" de la cookie son heredados de una clase específica que posee el paquete "org.apache.commons.httpclient" cuyo nombre es "NameValuePair".

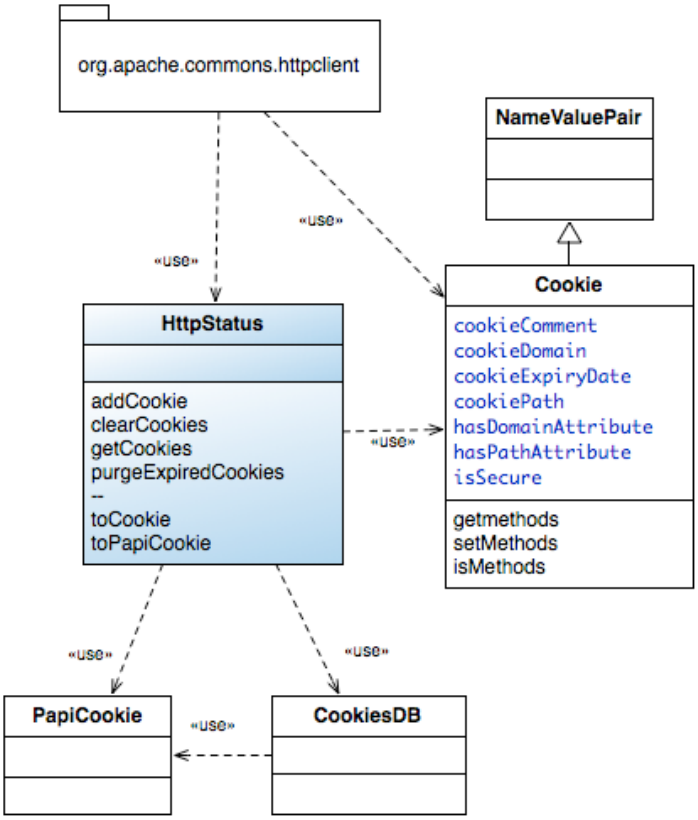

Fig. 3-16 *Diagrama de clases correspondiente a las clases de la librería Jakarta HttpClient relacionadas con la gestión de cookies.*

La segunda clase "HttpStatus", que también se muestra en la

Fig. 3-16, implementa una serie de métodos para el manejo de cookies: "addCookie", "clearCookies", "getCookies", "purgeExpiredCookies", suficientes para poder trabajar con un repositorio de cookies. A diferencia de RT-HTTPClient, con los métodos "requestHandler" y "trailerHandler", aquí no se ve afectado ningún método para el manejo de consultas o respuestas. Esto se debe a que HttpClient no utiliza dos repositorios de almacenamiento de cookies (memoria para temporales y otro permanente para cookies de larga duración), si no que utiliza un repositorio de cookies para todo, por lo que sobrescribiendo las sentencias de código de lectura, escritura, borrado y espiración de cookies, es suficiente.

Como resultado del análisis de la librería y las dependencias entre sus clases y con respecto a las cookies, la adaptación de la librería se ha realizado modificando exclusivamente la clase "HttpStatus", y más concretamente sus llamadas de: lectura, escritura, borrado, y expiración. Para mantener el código original de HttpClient lo más inalterado posible, se optó por dejar la clase "Cookie" intacta y se trasladó los métodos de conversión de cookies, entre Cookie y PapiCookie, necesarios para poder utilizar los métodos de CookiesDB a la nueva clase "HttpStatus". Como resultado, la nueva clase HttpStatus mantiene los mismos métodos que tenía y además incluye dos nuevos métodos de traducción de cookies: "toCookie" y "toPapiCookie".

Respecto al código modificado en los métodos antiguos, en las partes dónde se realizaba lectura y escritura de cookies, sólo se han realizado modificaciones respecto a cómo se lee y se escriben las cookies, forzando la lectura y escritura sobre el repositorio común de cookies implementado por papiCookieManager.

3.7.4. La integración de la clase HTTP estándar de JAVA

A partir de la versión 1.5 de JAVA se incluye el concepto de manejador de cookies "CookieHandler" para conexiones HTTP realizadas con las clases nativas de JAVA. La ventaja de esta integración es que es más mantenible que las otras dos que hemos visto hasta ahora (RT-HTTPClient y Jakarta HttpClient) y que resulta mucho más fácil adaptar a PAPI códigos JAVA ya existentes, ya que con incluir la llamada "setCookieHandler" y el nuevo handler a utilizar, es suficiente. La implementación del concepto "CookieHandler" lo realiza JAVA, tal y como muestra en la Fig. 3-17, a través de una clase abstracta, con ese mismo nombre, en la que deja dos métodos definidos y

listos para ser implementados por una clase hija. Los métodos en cuestión son: "get" y "put".

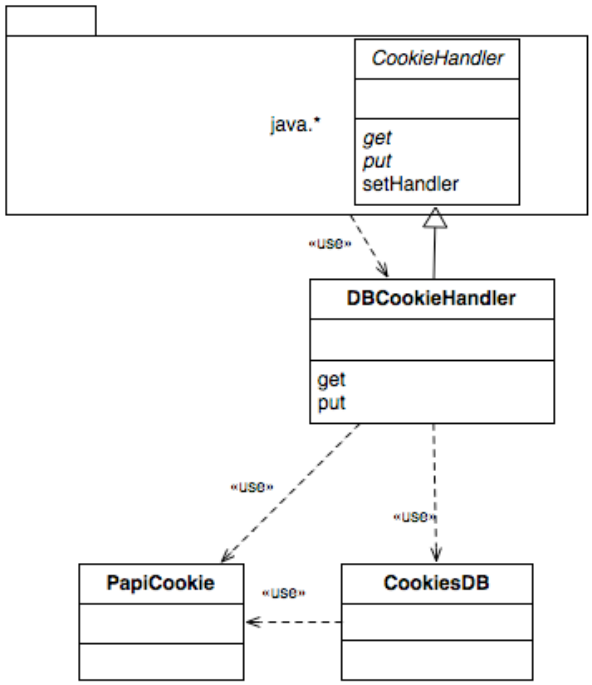

Fig. 3-17 *Diagrama de clases correspondiente a la implementación de la interfaz CookieHandler utilizando el repositorio de cookies PAPI para aplicaciones*

El método "GET" es llamado por JAVA justo antes de realizar una consulta HTTP. Al método se le pasa como parámetros; la URL de la consulta, y un conjunto de cabeceras HTTP que van a formar parte de ella. El método se encarga de, a partir de esta información, obtener las cookies que deben formar parte de la consulta e incluir las cabeceras "Cookie" que correspondan, en el conjunto de cabeceras que recibió como entrada.

El método "PUT" es llamado por JAVA justo después de recibir una respuesta a una consulta HTTP. Al método se le pasa como parámetros; la URL de la consulta y un conjunto de cabeceras HTTP que son enviadas en la respuesta. El método se encarga de, a partir de esta información, procesar las cabeceras "Set-Cookie" y almacenar las cookies que se estimen correctas.

Tal y como muestra la Fig. 3-17, en la solución se ha optado por implementar una clase "DBCookieHandler", que extiende la clase "CookieHandler" y que implementa los

métodos "get" y "put". En este caso, al no existir una clase Cookie previa, no ha sido necesario implementar métodos de traducción entre cookies. La clase "DBCookieHandler" trabaja con las cookies "PapiCookie" de forma nativa, y se incluye dentro del paquete "papiCookieManager".

3.8. Ampliación del sistema Single-Sign-On de PAPI

3.8.1. La tecnología JWS

En la implementación del entorno de participación remota, se ha utilizado Java Web Start como infraestructura para la ejecución de aplicaciones JAVA desde entornos web (desde el navegador), sin más que hacer acceder a un link enlazado a la aplicación a ejecutar. Además de ser una tecnología para lanzar de forma cómoda aplicaciones JAVA, incluye todo un conjunto de funcionalidades para control de versión y actualización de las mismas, así como de los recursos que éstas utilizan. JWS está incluida en la distribución básica de JAVA, que en su aplicación de configuración del entorno de ejecución, se incluyen parámetros y opciones relativos a JWS.

Java Web Start utiliza como tecnología base JNLP [29]. Mediante JNLP se puede definir todo un entorno de ejecución para una aplicación, en el que se incluye: resursos necesarios y su localización, opciones relativas a la seguridad en la ejecución, asignación de variables de entorno y propiedades de sistema, opciones relativos a la máquina virtual java, y parámetros de ejecución. La definición de este entorno de ejecución se realiza mediante la creación de un fichero con extensión "jnlp", con tipo mime "application/x-java-jnlp-file", y que puede estar localizado tanto en un servidor web, como en un dispositivo del equipo cliente del usuario.

En la Fig. 3-18 se muestra una estructura de un fichero JNLP. Como se puede observar, el documento comienza con un elemento cabecera "jnlp" en el que se definen los atributos: "spec" para describir la especificación a la que se ajusta el documento, y "codebase" que establece una raíz URL que será aplicable a todas las URLs relativas del documento. A continuación el documento jnlp se estructura en cuatro secciones:

"information": esta sección incluye información descriptiva de la aplicación a ejecutar, que puede resultar interesante, que no afecta al entorno de ejecución de la aplicación. Ejemplo de los datos que pueden aparecer en esta sección son: el título (title), la empresa u organismo distribuidor de la aplicación (vendor), una descripción (description), la pagina web con información de la aplicación (homepage), o el gráfico para utilizar como icono de la aplicación (icon).

"security": en esta sección se establecen las restricciones de seguridad que requiere la aplicación para su ejecución. Existen dos tipos de entorno de seguridad: "<all-permissions/>" y " <j2ee-application-client-permissions/>". En el primer caso, hay que darle acceso total a la aplicación (dentro del entorno del usuario que ejecuta la aplicación) a la máquina donde se va a ejecutar. En el segundo caso, el entorno de seguridad dispone de unas propiedades que limitan las acciones de las aplicaciones que se ejecutan. Para autorizar a una aplicación a disponer de recursos de acceso controlado, es necesario que el código que se va a descargar y ejecutar esté firmado digitalmente.

"resources": esta sección engloba aquellos recursos que la aplicación necesita para poderse ejecutar. Entre estos recursos están: la máquina virtual JAVA, librerías nativas (".os" o ".dll" por ejemplo), o paquetes de JAVA (ficheros "jar"). Para cada uno de estos recursos se puede fijar: la versión mínima del recurso requerida, la URL donde poder descargarlo, o el sistema operativo para el que está preparado dicho recurso (en referencia al sistema operativo de la máquina local donde se va a ejecutar la aplicación). Además en esta sección se pueden asignar valores a propiedades del sistema, que pueden ser consultadas dentro del código de la aplicación mediante sentencias "System.getProperty".

"application-desc": en esta sección se define la clase principal "main-class" donde la JVM intentará localizar primer método main a ejecutar. Además, en esta sección se pueden definir tantos argumentos (etiquetados como <argument>) como sea necesario.

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp spec="1.0+" codebase="http://www.mysite.com/application/" ...>
     <information>
          ...
     </information>
     <security>
          ...
     </security>
     <resources>
          ...
     </resources>
     <application-desc>
          ...
     </application-desc>
</jnlp>
```

Fig. 3-18 Estructura estándar de un fichero JNLP de la tecnología JWS

En el entorno de participación remota del TJ-II, las aplicaciones JAVA son lanzadas y se mantienen actualizadas en los clientes, mediante la tecnología JWS. La Fig. 3-19 muestra un ejemplo típico de fichero JNLP de una las aplicaciones del TJ-II. En este caso se trata de la aplicación eBoard utilizada por el piloto del experimento. A continuación se van a comentar brevemente las secciones del ejemplo:

- "information": se incluye información sobre la aplicación (el nombre y la página web donde obtener información adicional sobre ella) y el equipo de desarrollo.

- "security": se requiere permiso de acceso a todos los niveles que sea posible para el usuario que ejecuta la aplicación. Este tipo de seguridad requiere de la firma de todo el código que vaya a ser utilizado por la aplicación.

- "resources": se establece una versión mínima de JAVA igual o superior a "1.5". Además se requieren dos librerías: "eboard1.jar" (que contiene las clases desarrolladas para la aplicación) y "papiCookieManager.jar" que, como ya se ha comentado previamente, contiene toda la interfaz para uso del repositorio de cookies PAPI y que es recurso indispensable para cualquier aplicación JAVA integrada en PAPI.

- "application-desc" en la que se define la clase principal donde se inicia la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://fudaqs2.ciemat.es/TJ2WEB/" href="eboard1.jnlp">
    <information>
        <title>TJII Logbook App</title>
        <homepage href="eboard1.html"/>
        <vendor>TJ2WEB Data Acquisition Group</vendor>
        <offline/>
    </information>
    <security>
        <all-permissions/>
    </security>
    <resources>
        <j2se version="1.5+" />
        <jar href="bin/eboard1.jar"/>
        <jar href="bin/papiCookieManager.jar"/>
    </resources>
    <application-desc main-class="eboard1.Application1" />
</jnlp>
```

Fig. 3-19 *Ejemplo de fichero JNLP para lanzar la aplicación eBoard.*

Como se ha comentado previamente, para que una aplicación obtenga "all-permissions" a nivel de seguridad, JWS exige dos requisitos. El primero es que la aplicación y todos los ficheros JAR que utiliza estén debidamente firmados mediante el mismo certificado digital. El segundo requisito es que el usuario autorice la ejecución en base a la confianza del certificado con el que se ha firmado el código de la aplicación. Una vez que un usuario confía en el código firmado con un cierto certificado, puede establecerse una opción para que este usuario no vuelva a ser preguntado en futuras aplicaciones firmadas con dicho certificado.

La firma de una aplicación JAVA en el TJ-II se realiza mediante la aplicación "jarsigner" y utilizando un certificado autofirmado, generado específicamente para la firma de las aplicaciones JAVA utilizadas en el TJ-II. En el caso de las aplicaciones que se integran en PAPI [41], requieren de la librería "papiCookieManager.jar". La distribución del TJ-II de esta librería ya está firmada con el certificado antes mencionado, por lo que no requiere volver a ser firmado, y con encontrarse la línea: "<jar href="bin/papiCookieManager.jar"/> es suficiente.

La tecnología JWS más que ser un impedimento adicional cara a la integración de las aplicaciones JAVA en PAPI, ha resultado ser un gran aliado, ya que, por un lado, ha permitido, de forma transparente para el usuario, la automatización de procesos necesarios para el manejo de credenciales PAPI y la gestión del repositorio de cookies PAPI (por ejemplo en las fases de autenticación y "Logout"), y por otro lado ha permitido estructurar la integración de las aplicaciones JAVA en PAPI.

3.8.2. PAPI Cookies Loader

Una vez analizado el problema de la integración de aplicaciones JAVA con PAPI, y habiendo desarrollado una solución a través del repositorio común de "cookies" PAPI, (con sus diferentes interfaces y adaptaciones ya comentadas), el siguiente paso para una solución completa pasa por conseguir un sistema "Single-Sign-On" que englobe tanto al navegador web, como a las nuevas aplicaciones. Como ya se ha comentado anteriormente, una de las características más importantes de PAPI cara a su usabilidad, y a conseguir una solución útil para el usuario, es la de poseer un sistema que con una

sola autenticación, el usuario es identificado por todos los recursos a los que éste está autorizado a acceder. Además, y como otro de los requisitos principales de PAPI, la solución tiene que resultar lo más transparente como sea posible para el usuario.

Uno de los requisitos a la hora de encontrar una solución al problema antes descrito, es el de mantener el navegador web como software cliente con el que se realiza la autenticación, ya que sigue siendo la interfaz más estándar y natural para el usuario. Otro de los requisitos que se fijó es que todos los procesos implicados en la autenticación y que requirieran de interacción por parte del usuario, se circunscribieran en el entorno de la web de autenticación del usuario. Ya que una situación en la que un usuario ve aparecer mensajes y acciones relativas al proceso de autenticación, fuera de su web de autenticación, le lleva y le debe llevar a desconfiar de dicha acción. Mientras que si todos esos procesos e interacciones ocurren en su entorno natural de autenticación (normalmente el de su organización), la confianza del usuario es mayor.

Una vez analizado el problema, la solución se centró en la búsqueda de un mecanismo, en el entorno de la autenticación por web, que permita el manejo de repositorios de tokens de sesión, aparte del que el navegador web posee por defecto. A este mecanismo se le denomina "Loader", y en el caso del repositorio común de "cookies" JAVA "Cookies DB", se denomina "JAVA Cookie Loader". Y como tecnología base para su implementación, se optó por Java Web Start y su JAVA Network Launching Protocol, que permite establecer enlaces, de forma bastante transparente para el usuario, entre el protocolo HTTP y una aplicación (que funciona como "Loader"), y provee mecanismos de control para su ejecución.

En una primera fase se desarrolló una solución, en la que, por motivos prácticos se realizó una adaptación, para conseguir la solución final actualmente en funcionamiento. Por motivos de claridad se ha considerado más adecuado mostrar en un primer lugar la solución como fue concebida en una primera fase y posteriormente describir la modificación incluida junto con su motivación:

1. El usuario inicia una acción LOGIN contra el servidor de autenticación de su organización.
2. El AS comprueba si el usuario ya está autenticado, y en caso negativo, devuelve una página de autenticación para que usuario pueda proceder a identificarse.

3. El usuario a través del navegador web envía sus credenciales al SA.
4. Si la identificación es incorrecta, el SA devuelve una página web indicando el error en la identificación.
5. En caso de una identificación correcta, el SA verifica los recursos PoAs y GPoAs registrados para dicho usuario. Además, el servidor de autenticación comprueba para cada recurso, si tiene registrada como aplicación cliente, alguna que requiera el uso de un cargador.
6. De ser así, incluye en la lista de "sites" (recursos), en los que hay que cargar tokens de sesión ("cookies" PAPI), una nueva URI por cada tipo de cargador. En dichas URIs se indica la acción que la generó ("LOGIN" en este caso).
7. El servidor de autenticación devuelve al cliente una página web en la que se incluyen URIs de los sites, ya firmadas, para cada uno de los "sites" de los que tiene que precargar "cookies", así como una URI por cada cargador a lanzar en una segunda fase.
8. El navegador web del usuario resuelve en una página web de respuesta del SA y resuelve las URIs incluidas.
9. El servidor de autenticación, en una segunda fase, recibe una consulta HTTP por cada cargador, y genera como respuesta para cada una de ellas, un documento JNLP con información sobre el lanzamiento del cargador y parámetros de ejecución que indican, entre otras cosas, la acción a realizar y la lista de "sites" de los que se requiere cargar las "cookies" JAVA. Para cada uno de ellos, el SA genera una URI firmada.
10. El navegador web del usuario lanza la aplicación cargador que se indica en el JNLP recibido del SA.
11. La aplicación de carga "Loader", para cada URI firmada que recibe como entrada, realiza la carga de sus "cookies".
12. A partir de este punto todos los repositorios PAPI que pueda necesitar las aplicaciones que va a utilizar el usuario están listos.

En el caso de una acción LOGOUT, la secuencia es equivalente.

Como ya se ha comentado previamente, como consecuencia de la experiencia de los usuarios y para mejorar la usabilidad del sistema, se realizó una modificación sobre la primera solución. Los usuarios solicitaron poder cargar las credenciales (en este caso para aplicaciones JAVA) a demanda, es decir, en vez de que el cargador de credenciales JAVA se lanzara por defecto como respuesta a una autenticación satisfactoria, solicitaron disponer de un enlace en su página de autenticación (igual que el botón de "logout") para poder cargar las credenciales en caso de necesitarlo. En este caso la secuencia equivalente del proceso es prácticamente igual, salvo que en el paso 7, la URI correspondiente al cargador JAVA, no se incluye en un recurso que el navegador intente cargar de forma automática, si no que se incluye como un enlace que requiere de la acción del usuario. De igual forma, cuando se realiza LOGOUT, sólo se incluye una URI correspondiente a la acción LOGOUT en el cargador JAVA, si y sólo si, hay registro de que el usuario solicitó la carga de credenciales JAVA.

3.8.2.1. JAVA CookiesDB Loader

El caso del cargador JAVA CookiesDB es una implementación de un "PAPI Loader", cuya función es gestionar el repositorio común de "cookies" PAPI para aplicaciones JAVA. Se utiliza en las fases de "loggin" y "logout", para precargar y borrar respectivamente las cookies del repositorio. Se basa en la clase "CookieLoader" que implementa una serie de métodos relacionados con la gestión del repositorio de "cookies" JAVA. Para ello, hace uso de las librerías "CookiesDB", "PapiCookie", y de la librería RT-HTTPClient adaptada a PAPI.

Como "Loader" de PAPI, utiliza la tecnología JWS para automatizar y hacer lo más transparente posible al usuario la carga de "cookies". La Fig. 3-20 muestra un JNLP típico asociado al JAVA "Loader". En éste se puede observar las típicas secciones (ya comentadas): "information", "security" o "resources" y "application-desc". Este JNLP es igual para todas las cargas de credenciales JAVA. El servidor de autenticación lo genera a partir de una plantilla en la que queda por rellenar la lista de argumentos que se corresponden con URIs firmadas. Como recursos, el JAVA "Loader" sólo requiere de la librería "papiCookieManager" en la que se incluye: todo lo necesario para utilizar el repositorio de "cookies", el código de la clase CookieLoader, así como una versión completa de la librería RT-HTTPClient adaptada a PAPI. Por facilitar la distribución, se han incluido todas las clases utilizadas por PAPI en un solo fichero JAR.

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="http://fudaqs2.ciemat.es/PAPI/Loaders/java">
 <information>
        <title>PAPI Token Loader</title>
        <vendor>TJII Acquisition Group (CIEMAT)</vendor>
        <description>Program for loading PAPI tokens</description>
 </information>
 <security>
        <all-permissions/>
 </security>
 <resources>
        <j2se version="1.5+"/>
        <jar href="papiCookieManager.jar" />
 </resources>
 <application-desc main-class="papiUtils.CookieLoader">
       <argument>LOGIN</argument>
       <argument>https://efdafed.fusion.ciemat.es/efda_fed_GPoA/PAPI/cookie_ha
ndler.cgi?AS=CIEMAT_AS&ACTION=LOGIN&DATA=jRvpwDezRwXQysNvibz1FLADYHg7%2Fh388b0
FzSUyOO8bh3D5UaanMB6607jMARsiAHs9QP13BiaEOiRWitq3ccTUcY6k%2F0wEBEO4UhuwJL7bkW4
JWOUzwKzGN4qo1pZbX6eGAKNVO4NzlHXvdVVUZYRupOKBIA95GE12K%2Fz7aeNGWdV34eJCc5lLfkr
eYo%2Fkggr5ZG4xTfZj1NTSEbsSv0uZVYWMQ8bsLQpggJHbHsqaC72Ix%2FQ3bac5rwdtf%2FGB8Ag
1l9XuLpoIt80ZDG02VcOWkLkrrbgTtE4DejfoZOFYe3XFqlbrdMt1Cd8ILlnVGvEYvdMj8uCni73pl
XZnAA%3D%3D</argument>
       <argument>http://fudaqs2.ciemat.es/TJ2WEB/APP_JAVA/PAPI/cookie_handler.
cgi?AS=CIEMAT_AS&ACTION=LOGIN&DATA=e51Ue5X%2Bi%2Bxn1DMGmtA31h43QcOs4JW3g6%2FOT
5SNyFDClPHnTDlNGbD10%2FB2HB2uCJqVPEn8ax8D%2B8SdWytzzFlYo6xpXvc%2FkS66US1s40yqd
sclAuIq5EDZSClYt30wJTPuOb1DLAG8w214HdJ9CsoVU6JKTO98OFhL3jMKUeY%3D</argument>
 </application-desc>
</jnlp>
```

Fig. 3-20 *Fichero JNLP generado por el servidor de autenticación como respuesta a una petición de carga de credenciales para aplicaciones JAVA*

Con la idea de conservar el sistema de carga de "cookies" de PAPI y mantener la solución lo más compatible que sea posible, el método de carga de "cookies" que utiliza el "Loader" de JAVA es equivalente al que realiza el navegador web. Es decir, si la acción es LOGIN, para cada una de las URLs firmadas que recibe como entrada, realiza una consulta HTTP con dicha URI. Al otro lado hay PoAs recibiendo la consulta y devolviendo "cookies" PAPI cifradas para ser cargadas. Al utilizar RT-HTTPClient adaptada a PAPI como librería HTTP para realizar las consultas, las "cookies" respuesta son correctamente incluidas en el repositorio. Además, gracias a esta librería, también pueden ser referenciados recursos HTTPS.

En el caso de LOGOUT se realiza una limpieza de "cookies" PAPI en el repositorio y se bloquea el acceso de nuevas "cookies" PAPI (para ese usuario) hasta un nuevo

LOGIN. Esto se hace para evitar que se carguen de forma errónea "cookies" que provienen de respuestas posteriores al LOGOUT.

Además de las funcionalidades propias de LOGIN y LOGOUT, se han implementados métodos que pueden ser utilizados desde línea de comandos y ayudan, tanto a gestionar el repositorio de forma manual, como a realizar funciones de depuración:

- **add:** Permite añadir una entrada en el repositorio de "cookies" PAPI, a partir de los siguientes parámetros típicos en una cookie: dominio, nombre, valor, path, y fecha de expiración.

- **del**: Permite eliminar una cookie del repositorio a partir de su dominio, nombre, y path.

- **print**: Muestra por pantalla la lista de las "cookies" actuales en el repositorio.

3.8.3. PAPI Runner

Una de las principales preocupaciones desde el comienzo del proyecto ha sido conseguir una buena integración de las aplicaciones JAVA con PAPI. La inclusión de un mecanismo de soporte de "cookies" "CookieHandler" a partir de la versión de JAVA 1.5, supuso un gran avance en este sentido y abría muchas más posibilidades a la hora de integrar aplicaciones sin necesidad de recompilar su código JAVA. Uno de los aspectos más importantes que introduce CookieHandler es la posibilidad de aislar completamente el manejo y almacenamiento de "cookies", del resto de la aplicación. Para ello, CookieHandler utiliza el concepto "Hook", que se basa en dotar a un sistema de un punto de "anclaje" que permita añadir un módulo para realizar una función concreta, sin necesidad de tocar la implementación del resto del sistema. Como consecuencia, el sistema y el “Hook” están altamente desacoplados en su implementación, y por tanto resulta muy fácil cambiar el "Hook" sin que el resto del sistema se vea alterado. La clase CookieHandler se utiliza en JAVA como método de enlace a un subsistema de manejo de "cookies" completamente desacoplado del resto de la aplicación. Como interfaz con el subsistema de manejo de "cookies", se utilizan dos métodos abstractos "put" y "get" a implementar y que han sido descritos en el punto anterior. Estos métodos son llamados por JAVA justo antes de realizar una consulta

HTTP (get) y justo al recibir la respuesta (put) y por tanto el resto del sistema no sabe ni de "cookies" ni de repositorios.

A partir de la implementación de la clase DBCookieHandler, incluida en el paquete papiCookieManager y capaz de manejar "cookies" PAPI, se ha desarrollado un mecanismo capaz de integrar dicho manejador en aplicaciones JAVA, sin necesidad de ser recompiladas. Este mecanismo, llamado PAPIRunner [41], tiene la función principal de lanzar la aplicación JAVA que se le pasa como parámetro, y asignar DBCookieHandler como manejador de "cookies" por defecto para dicha aplicación.

Otra funcionalidad incluida en PAPIRunner tiene que ver con un fallo que se ha detectado en JAVA versión 1.5 respecto a la implementación de las llamadas a CookieHandler en los procesos de redirección. En caso de consultas HTTP comunes, los métodos "get" y put" de la clase CookieHandler activa, son llamados en cada consulta y respuesta. Sin embargo, en cuanto existe una redirección HTTP (30X), las siguientes consultas HTTP relacionadas con dicha redirección, ya no utilizan CookieHandler, ni sus métodos "get" y "put" son utilizados. Este problema hace que los mecanismos de redirección requeridos por el mecanismo GPoA de PAPI resulten no operativos. Para solucionar este problema, PAPIRunner realiza una primera precarga de credenciales para los recursos a los que la aplicación va a acceder. Para ello utiliza una de las librerías HTTP adaptadas a JAVA y que sí manejan bien las redirecciones HTTP (en concreto RT-HTTPClient) para realizar la precarga de "cookies" válidas.

La

Fig. 3-21 representa un gráfico en el que se observa las diferentes fases por las que pasa PAPIRunner para lanzar una aplicación.

Fase 1: se inicia PAPI Runner y se establece DBCookieHandler (gestor de "cookies" PAPI) como manejador de "cookies" por defecto.

Fase 2: PAPIRunner comprueba el valor de la propiedad "PapiURLs" que contiene la lista de las URLs correspondientes a los recursos que accederá la aplicación. PAPIRunner recorre todas estas URLs y para cada una de ellas genera una consulta HTTP utilizando la librería RT-HTTPClient adaptada a PAPI. Con ello se garantizan dos cosas. Por un lado que todas las "cookies" PAPI van a poder ser precargadas, ya que si existe algún recurso enlazado a un GPoA, la redirección que es necesaria hacer para poder identificar al usuario, va a ser realizada con una librería que maneja

perfectamente las redirecciones. Por otro lado, verificar en el momento del arranque si la aplicación tiene permiso para acceder a todos los recursos necesarios, ya que si existe algún recurso sin acceso, en esta fase es detectado. De darse el caso, la librería papiCookieManager incluye una solución (configurable) que muestra, al comienzo de la aplicación, un aviso al usuario comunicando el problema y posibles soluciones.

Fase 3: una vez todas las credenciales necesarias están cargadas, PAPIRunner carga la aplicación a ejecutar (que viene definida en el primer argumento de ejecución) y establece DBCookieHandler (preparado para "cookies" PAPI) como manejador de "cookies" por defecto.

Fase 4: Finalizada la labor de PAPIRunner, este deja todo el control de la ejecución a la aplicación cargada, es decir, la aplicación que se quiere ejecutar y que fue previamente cargada en la fase 3.

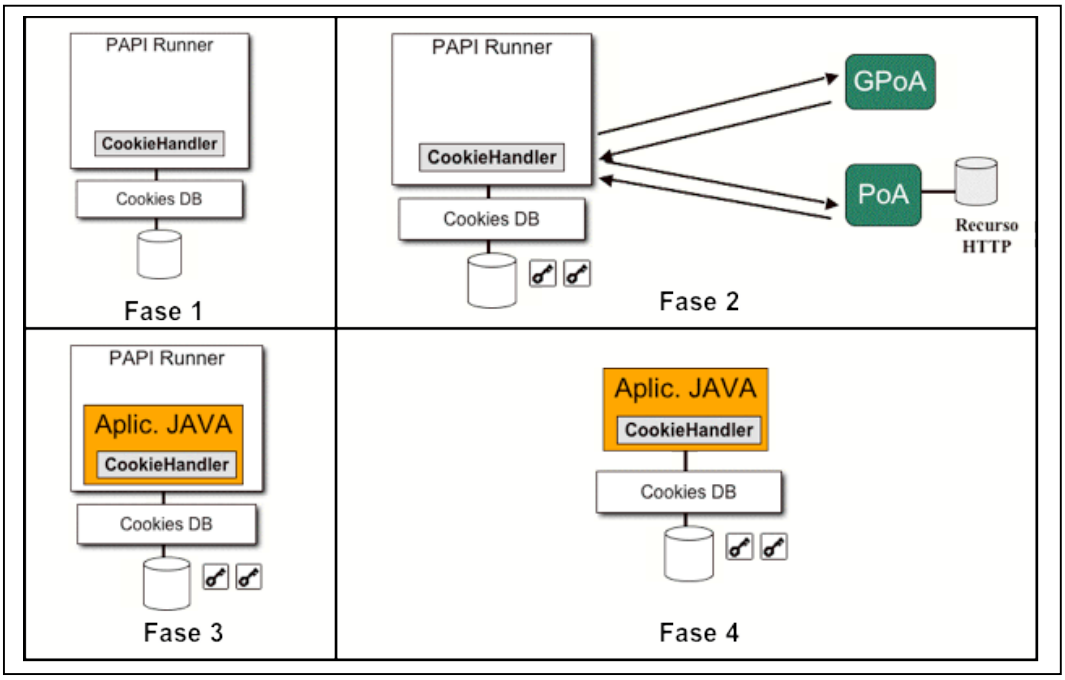

Fig. 3-21 *Diagrama de las fases por las que pasa la ejecución de una aplicación JAVA utilizando PAPI Runner.*

Capítulo 4

Implantación de PAPI como infraestructura de seguridad multi-organización para EFDA

4.1. Descripción del problema

Las investigaciones en energía de fusión en Europa son realizadas en una serie de laboratorios y centros pertenecientes a diferentes países. EFDA es una organización orientada a potenciar la colaboración y coordinación entre estos centros. En cada laboratorio de fusión existe una unidad de control y adquisición de datos, responsable de gestionar los sistemas de control de la máquina de fusión de ese laboratorio, además de gestionar la adquisición, almacenamiento y recuperación de los datos generados en los experimentos realizados en dicha máquina. Los investigadores analizan constantemente los datos recogidos de diferentes experimentos. Para ello, las unidades de control y adquisición desarrollan, y ponen a disposición de los investigadores, herramientas que facilitan la recuperación y visualización de los datos almacenados.

Las máquinas de fusión en los diferentes laboratorios son diferentes y es frecuente que investigadores de un centro trabajen con los datos obtenidos en otros laboratorios. Dentro de EFDA se encuentra JET (Joint European Torus). Esta instalación destaca dentro de las organizaciones pertenecientes a EFDA por el tamaño de su máquina de fusión y por ser modelo para el desarrollo de la futura máquina de fusión a mayor escala englobada dentro del proyecto internacional ITER al que pertenecen diferentes países. JET es un laboratorio de referencia dentro de la comunidad internacional de fusión, de ahí que los investigadores realicen y contrasten continuamente sus investigaciones sobre los datos y experimentos realizados en JET. Por otra parte, dentro de EFDA existen

grupos de trabajo y coordinación, formados por miembros de los diferentes laboratorios, en los que se realizan proyectos en colaboración y en los que las herramientas de trabajo en grupo resultan esenciales. Todo ello hace de éste, un entorno perfecto para el desarrollo de tecnologías que faciliten y potencien el trabajo en grupo, así como el acceso remoto a datos y aplicaciones que se encuentran distribuidos por diferentes organizaciones. De la misma forma, ITER (Internacional Thermonuclear Experimental Reactor) nace con la filosofía de ser un laboratorio abierto a los investigadores de diferentes países y se busca potenciar el acceso remoto a datos y aplicaciones.

A nivel de seguridad, es necesario que el acceso a las diferentes aplicaciones y datos quede restringido a aquellos usuarios que estén autorizados a utilizarlos. Para ello se deben poder establecer políticas de acceso basadas en criterios coherentes con el entorno de laboratorios de fusión. A su vez, se debe minimizar el número de claves diferentes que un usuario debe conocer, así como la cantidad de veces que se debe autenticar para acceder a los diferentes recursos. De esta forma aumenta la sensación de entorno de trabajo remoto, permitiendo con una sola autenticación, el acceso a aquellos recursos a los que el usuario esté autorizado.

A nivel funcional, se debe maximizar el conjunto de aplicaciones comúnmente utilizadas. Cabe destacar en este sentido, que navegadores web como: Netscape, Mozilla, MS Explorer u Opera, son ampliamente utilizados por los investigadores. Así mismo, la tecnología JWS es ampliamente utilizada como mecanismo que facilita a los usuarios la instalación y actualización de sus aplicaciones.

Otro aspecto fundamental en este entorno de trabajo es la movilidad. Los investigadores tienen que moverse con cierta frecuencia de su lugar habitual de trabajo y es imprescindible para ellos poder seguir accediendo a los recursos a los que habitualmente están autorizados.

4.2. La Federación como estructura multi-organización

Cuando se aplica un sistema de autenticación y autorización a una estructura multi-organizativa, aunque dicho sistema resuelve aspectos técnicos para poder llevar a cabo un control de acceso adecuado, queda por cubrir una serie de aspectos (de carácter más organizativo y de coordinación) que resultan fundamentales para alcanzar una solución satisfactoria. Para poder cubrir estos elementos, se crea una estructura organizativa, que

engloba a las organizaciones implicadas y en la que se engloban y definen una serie conjunto de políticas, procedimientos y criterios que terminan a definir la solución final.

4.2.1. Características de un modelo federado

Identidad y privacidad: Este es un criterio básico de un modelo federado. Se trata de que se puedan asociar identidades a usuarios, recursos o componentes del sistema, y dichas identidades sean reconocidas por los diferentes elementos que forman parte de la federación y sirvan para poder identificar de forma unívoca a dichos elementos. Por ejemplo, a un usuario mediante un proceso de autenticación se le asocia una identidad y ésta le servirá para ser identificado frente a recursos u otros elementos de la federación.

Dentro de la federación, las organizaciones pueden jugar principalmente dos roles:

Proveedor de Identidad: Es el caso de una organización que gestiona sus usuarios y se responsabiliza de su autenticación. También puede mantener un repositorio con información complementaria sobre el usuario, que suele resultar útil para el acceso a recursos cuya política de filtrado requieren de dicha información.

Proveedor de Servicios: Se trata de organizaciones que ofertan servicios a usuarios pertenecientes a organizaciones de la federación. Estos servicios, normalmente, poseen una política de acceso o filtrado que se implementa mediante una serie de reglas de acceso.

La identidad dentro de la federación debe ser válida dentro de ésta y no necesariamente hay que asociar a un usuario una identidad que vulnere su privacidad. Dentro de una federación pueden manejarse identificadores que resultan válidos como identidad dentro de la federación, pero que sólo el proveedor de identidad puede asociar al usuario real. De esta forma, se protege la privacidad del usuario y a la vez, ante un caso de abuso en un servicio, se puede trazar el usuario que realizó dicho acceso.

Confianza: Este es otro criterio básico en un modelo federado. Deben existir relaciones de confianza a varios niveles:

- entre organizaciones: debe existir confianza a la hora de enviar datos sobre usuarios (por ejemplo que sean necesarios para implementar una cierta regla de control de acceso), y sobre el uso que la otra organización hará de ellos. Así mismo debe existir

confianza a la hora de recibir información desde otra organización, en el sentido de que la información recibida se considere veraz.

- entre elementos del sistema: en el sentido de que un elemento debe confiar en que cuando establece una comunicación con un segundo elemento, éste es quien dice ser y además la información intercambiada es veraz e íntegra.

Coherencia de atributos: Debe existir una coordinación entre las diferentes organizaciones que conforman la federación para conseguir que la información que una organización posee sobre un usuario es coherente a nivel sintáctico y semántico con las políticas de acceso que implementan los proveedores de servicio dentro de la federación. Por ejemplo, si un recurso posee una regla de filtrado tipo "nivel de seguridad > 3", debe existir una coherencia, tanto a nivel sintáctico como semántico, del atributo "nivel de seguridad" entre las organizaciones cuyos usuarios quieren acceder a este recurso. Para conseguir esto se acuerdan políticas de atributos que permitan esta coordinación.

Criterio de buen uso: Es recomendable para mantener la confianza entre los usuarios, proveedores de identidad y los proveedores de servicio, que exista un conjunto mínimo de reglas referidas al uso de la federación. Por ejemplo, mecanismos de autenticación permitidos, utilización de ciertos servicios, tipos de dispositivos autorizados, criterios para preservar la información relativa a los usuarios.

Criterio de incorporación: Es recomendable que se concrete una serie de criterios y procedimientos para la incorporación de nuevos miembros a la federación, tanto nuevos proveedores de identidad, como nuevos proveedores de servicio. Esto permite fortalecer la confianza entre las diferentes entidades de la federación, y facilita la gestión de la federación.

4.2.2. Elementos englobados en una federación

El concepto de federación dentro de un sistema de control de acceso que engloba a usuarios y recursos de varios centros, se basa en la idea de crear una estructura a nivel organizativo que coordine una serie de aspectos:

Política de atributos: las reglas de control de acceso con las que implementa una política de control de acceso a un recurso puede contener propiedades referentes al usuario que está intentado acceder. A estas propiedades del usuario: edad, cargo, etc.,

se las llama atributos del usuario. Los atributos del usuario suelen estar almacenados en un repositorio que normalmente coincide con el utilizado para realizar la autenticación. Para que las reglas de control de acceso tengan sentido, sus atributos tienen que coincidir en nombre, sintaxis y semántica, con aquellos almacenados en los de los repositorios de usuario. Para conseguir esta coherencia hay que establecer una política común de atributos que maximice la coordinación entre repositorios de usuario y políticas de control de acceso.

Política de adhesión: Se debe establecer un conjunto de reglas que regulen qué organizaciones o bajo qué procedimiento una nueva organización puede ser admitida en la federación. Así mismo se pueden regular otros aspectos como tipo de usuarios, tipo de recursos, etc.

Infraestructura de autenticación y autorización: Debe existir una solución tecnológica que soporte las funciones de autenticación y autorización para todos los usuarios y recursos de las organizaciones federadas.

Estructura de confianza: Es necesario implementar una estructura que posibilite la confianza entre los diferentes elementos del sistema que soporta la federación, es decir, que diferentes elementos que conforman la federación crean unos en otros. Para ello, cuando un elemento recibe información de otro, el primero puede autenticar al emisor y comprobar la integridad de los datos recibidos, y por supuesto, asume que la información que el otro le envía es veraz.

La confianza entre dos elementos del sistema se implementa normalmente en base a una tecnología de cifrado con clave, bien simétrica, bien asimétrica. En el primer caso (tipo algoritmo AES) la clave con la que se cifra la información es secreta y debe ser conocida únicamente por los dos elementos entre los que se establece la relación de confianza. En el segundo caso, se utiliza un par de claves (privada y pública) de tal forma que la clave privada es conocida únicamente por uno de los elementos del sistema y la parte pública por todos los otros. Cuando en el sistema a considerar intervienen varios elementos, con la primera técnica habría que crear una clave por cada relación de confianza entre 2 elementos, mientras que con la segunda habría que crear 2 claves por cada elemento del sistema. Con la técnica de clave asimétrica o clave pública se simplifica en gran medida el número de claves necesarias, por lo que es la comúnmente utilizada para este tipo de propósitos.

Centro repositorio: Se trata de un elemento que coordine los recursos disponibles dentro de la federación. Para cada uno de los recursos, es interesante conocer: su localización (URL), organización propietaria, aplicación cliente necesaria para su utilización, así como información referente a los atributos del usuario necesarios para acceder a dicho recurso.

4.3. Tecnologías alternativas a PAPI para implementar la federación

4.3.1. OpenID

OpenID [44] es un sistema que implementa un esquema de identificación distribuido, de tal forma, que un recurso puede identificar a un usuario en base a una autenticación que realiza un proveedor de identidad en una localización diferente. OpenID gracias a su gran sencillez a nivel de protocolo, concepción e implementación, está expandiéndose a gran velocidad entre proveedores de servicio que engloban grana cantidad de usuarios en Internet. Básicamente, estos proveedores se convierten en proveedores de identidad OpenID, de tal forma, que cuando alguno de ellos es preguntado, por un recurso, sobre la identidad de un usuario (utilizando el protocolo OpenID correspondiente), éste responde comunicando al recurso la identidad del usuario, o al menos, comunicándole si dicho usuario es reconocido por él.

La tecnología OpenID está orientada a la utilización del navegador web como aplicación cliente, aunque puede ser utilizada en aplicaciones desarrolladas de forma compatible con el modo de funcionamiento de OpenID y que provean al usuario de mecanismos de interacción necesarios.

La alta velocidad de expansión de este esquema, le convierten en un potencial estándar de facto y hacen de él un candidato a tener en cuenta en la implementación de una infraestructura de autenticación y autorización entre organizaciones.

4.3.1.1. Arquitectura de OpenID

El esquema OpenID lo conforman un conjunto de componentes que pueden estar localizados de forma distribuida en Internet. Tal y como muestra el diagrama de secuencia de un sistema OpenID, representado en la *Fig. 4-1*, estos componentes son:

- Propietario del recurso: Representa al recurso web a proteger. Este componente requiere del esquema OpenID una identificación del usuario que está intentando acceder a dicho recurso.

- Web de identificación: Esta web se corresponde con la URL que el usuario provee al propietario del recurso para que, a partir de aquí, éste comience todos los pasos que finalicen en una positiva identificación del usuario.

- Proveedor OpenID: Servidor encargado de comprobar si el usuario está correctamente autenticado y de comunicar de forma fiable este hecho al propietario del recurso.

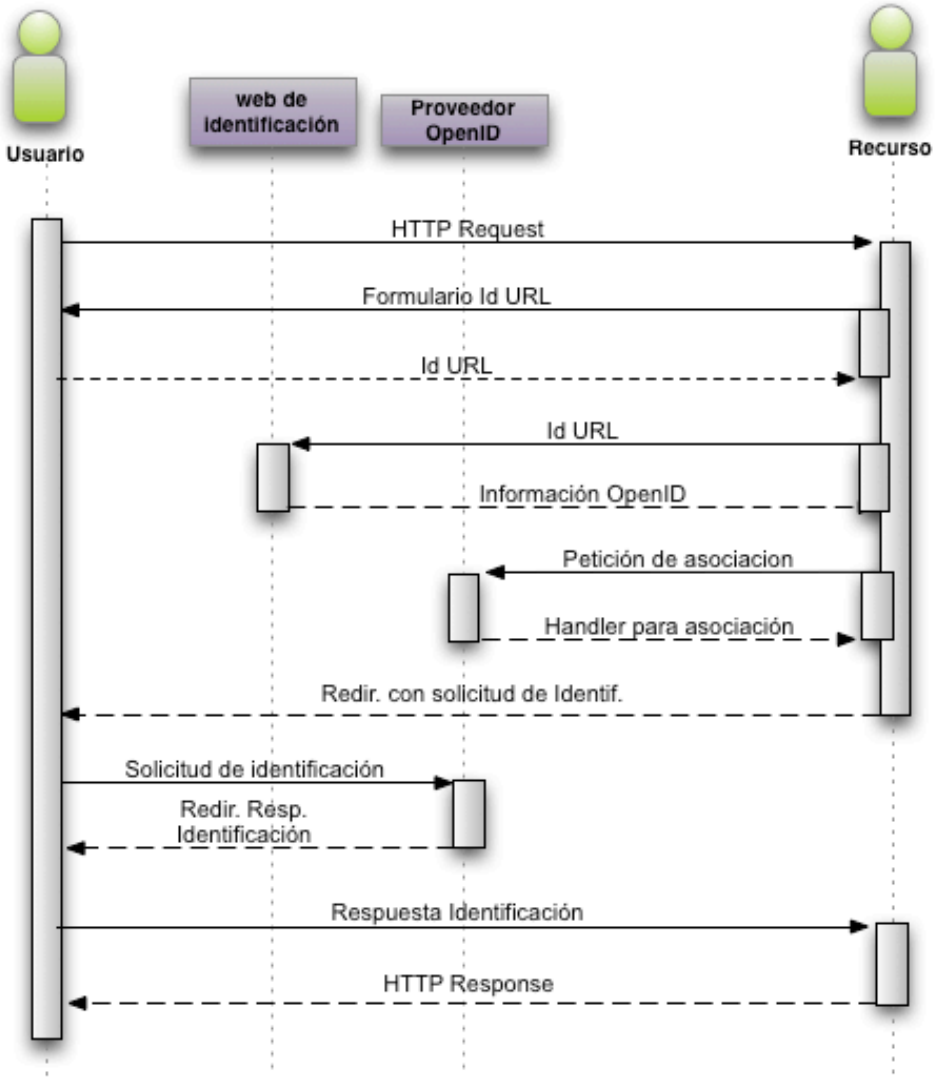

Fig. 4-1 *Diagrama de secuencia del esquema de identificación utilizado por OpenID*

La secuencia de todo es proceso es el siguiente:

1- El usuario a través de su aplicación cliente intenta acceder a un recurso web.

2- El propietario del recurso no es capaz de identificar al usuario y se solicita una URL de identificación de OpenID.

3- El usuario le envía la URL solicitada

4,5- El propietario del recurso, se comunica con la URL que corresponde con la web identificadora del usuario y obtiene de ella la dirección del proveedor OpenID del usuario. Para ello el cliente debe especificar este parámetro en una entrada tipo "link" y con valor de la propiedad "rel" igual a "openid.server" dentro de la sección "HEAD" de la página web. For example: <link rel="openid.server" href="http://openidserver.domain.name">

6,7- El propietario del recurso se pone en contacto con el proveedor OpenID y obtiene un handler que contiene una clave compartida para asegurar la integridad del mensaje en el proceso de identificación. El intercambio del handler se puede realizar utilizando una conexión segura y mecanismos de intercambio de clave tipo "Diffie-Hellman".

Consulta

- **openid.mode**: Campo que especifica el tipo de mensaje. En este acaso su valor es "associate".
- **openid.assoc_type**: Campo que define el tipo de asociación que se va a establecer entre el propietario del recurso y el proveedor OpenID. Por defecto se suele usar un método de firma "HMAC-SHA1" que combina una función HASH del mensaje para obtener un código de firma, junto con el cifrado de la firma obtenida. Es decir un método de HASH criptográfico. Se suelen utilizar "HMAC-SHA1" o "HMAC-MD5" en función del algoritmo de firma utilizado.
- **openid.session_type**: Especifica el modo de intercambio de clave compartida. Aunque este campo por defecto está vacío (lo que implica un formato de texto en claro), OpenID recomienda Diffie-Hellman con sistema de firma SHA1
- **openid.dh***: Conjunto de parámetros asociados a algoritmo de Diffie-Hellman elegido, siempre que este sea el método elegido en el campo "openid.session_type".

Respuesta

- **assoc_type**: Campo que define el tipo de asociación que ha definido entre el propietario del recurso y el proveedor OpenID. Por defecto se suele usar un método de firma "HMAC-SHA1" tal y como se ha descrito anteriormente.
- **assoc_handle**: El handler de la asociación
- **expires_in**: Número de segundos de vida para este handler. Una vez expirado el tiempo de vida, el handler no debe ser utilizado.
- **session_type**: El método de intercambio de handler elegido. Un método Diffie-Hellman es recomendable.
- **dh_***: Parámetros asociados al método de intercambio elegido.

8- El propietario del recurso redirige al usuario mediante redirección HTTP al proveedor OpenID solicitando su identificación. El mensaje incluido en la redirección incluye los siguientes campos:

- **openid.mode**: Acción solicitada al proveedor OpenID. En este caso "checkid_immediate".
- **openid.identity**: Identidad provista por el usuario para ser identificado por el proveedor OpenID.
- **openid.assoc_handle**: Handler de la asociación entre el propietario del recurso y el proveedor OpenID en el paso 6 y 7.
- **openid.return_to**: URL donde el proveedor OpenID debe redirigir al usuario con la respuesta.

9- El proveedor OpenID realiza la autenticación.

10- El proveedor OpenID devuelve al usuario a la web de recurso con información sobre su identidad firmada utilizando el método de asociación acordado en el paso 6 y 7 y cuyo handler también fue establecido en dichos pasos. Los campos que contiene son:

- **openid.identity**: Identidad del usuario verificada por el proveedor OpenID.
- **openid.assoc_handle**: handler cifrado para el cálculo de la firma del mensaje.

- **openid.return_to**: Copia del parámetro enviado en el paso 8.
- **openid.signed**: Lista de campos del mensaje (separados por coma), que están contenidos en la firma.
- **openid.sig**: valor de la firma del mensaje a comprobar por el propietario del recurso para comprobar la autenticidad e integridad del mensaje.

El sistema en general aporta ideas interesantes como el mecanismo de petición de asociación que evita el tener que utilizar un esquema de confianza basado, por ejemplo, en clave pública. Además su sencillez e integración en cualquier tipo de navegador justifica su gran velocidad de expansión. Aún así tiene una serie de inconvenientes importantes que hay que remarcar:

* No define ningún mecanismo ni sistema de mantenimiento de sesión y se deja al desarrollador esta tarea.

* No define ningún mecanismo de logout.

* El único modo de garantizar que una vez autenticado el usuario no requiera intervención adicional es mediante un proveedor OpenID único para toda la infraesructura. De lo contrario, al usuario se le solicitara la URL de identificación.

* No implementa mecanismos de autorización, se deja en manos del desarrollador.

* Sólo se ha definido para navegadores web. Y mientras no se consiga un autentico mecanismo de Sigle-Sign-On que evite la interacción con el usaurio, será muy difícil conseguir extender el sistema a otras aplicaciones.

En resumidas cuentas, aunque dispone de alguna solución interesante, realmente sólo resuelve parte del problema de la identificación del usuario y deja sin desarrollar el resto de aspectos funcionales de una infraestructura de autenticación y autorización.

4.3.2. Shibboleth

Shibboleth [45] nace como uno de los proyectos englobados en la iniciativa Internet2. El objetivo principal de este proyecto consiste en el desarrollo de un sistema single sing on para entorno web y que funcione en entornos multi-organizazión. Con ello se pretende conseguir accesos autorizados a recursos protegidos, preservando la privacidad.

Shibboleth está basado en la especificación SAML [46] (Security Assertion Markup Language) v1.1 desarrollada por el grupo OASIS (Organization for the Advancement of Structured Information Standards), y sirve de base para crear estructuras federadas entre organizaciones, permitiendo la transmisión de información relativa a los usuarios que posibilite políticas de acceso basadas en dicha información. Gracias a los mecanismos de gestión de identidad única, Shibboleth simplifica la gestión de usuarios y reglas de control de acceso entre las organizaciones federadas.

4.3.2.1. Arquitectura Shibboleth

En Shibboleth, cuando un usuario, mediante un navegador, intenta acceder a un recurso protegido, el servidor del recurso notifica que no tiene atributos del usuario para poder decidir si dejarle pasar. A partir de aquí, y como muestra la Fig. 4-2, el SHAR (Shibboleth Attribute Requester), interactuará con la AA (Attribute Authority) de la organización origen del usuario (aquella a la cual pertenece el usuario dentro de la federación), para solicitarle los atributos necesarios. La autoridad de atributos obtendrá los atributos del usuario, bien directamente de un repositorio local, o bien de algún servidor LDAP o base de datos de usuarios de esa organización.

Para poder decidir si puede enviar dicha información del usuario a la organización que lo demanda, para acceder al recurso en cuestión, la AA hace uso de la ARP (attribute release policy) del usaurio. La consulta que el SHAR envía a la AA, se denomina "AQM" (attribute query message), y la respuesta se denomina "ARM" (attribute response message).

Una vez el SHAR del servidor a obtenido los atributos necesarios, se los pasa al RM (Resource Manager) que los solicitó, para que este aplique la política de acceso que corresponda, permitiendo o denegando el acceso.

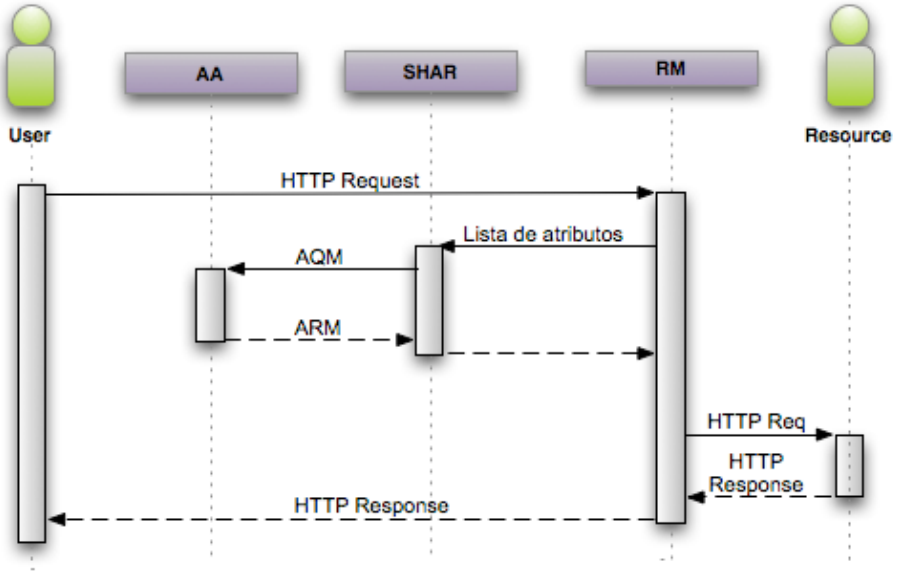

Fig. 4-2 *Diagrama de secuencia del protocolo SHAR - AA en Shibboleth*

Como se puede ver en la Fig. 4-3 los diferentes componentes del sistema shibboleth se reparten entre la organización origen del usuario, la propietaria del servicio (organización destino) y hay elementos comunes a la federación, como WAYF que pueden estar alojados en terceras entidades.

4.3.2.2. Shibboleth Handler

Por un lado Shibboleth quiere mantener un sistema single-sing-on, y por otro lado es un sistema concebido para trabajar en entornos de varias organizaciones, por lo que debe manejar un sistema de identificación común que permita identificar al usuario. Así mismo debe mantener privacidad, por lo que la auténtica identidad del usuario debe poder ser preservada.

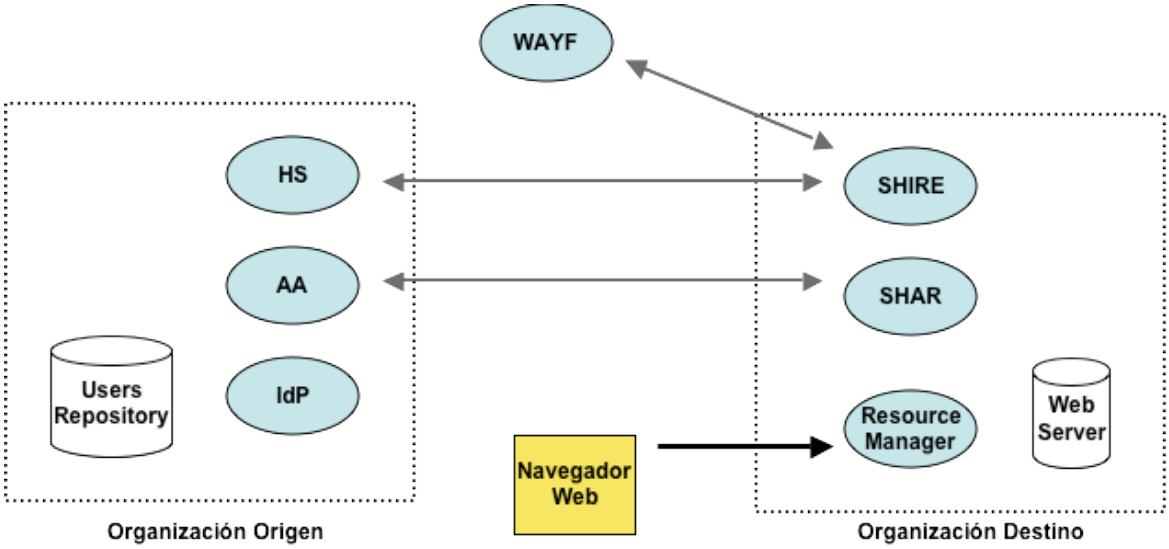

Fig. 4-3 *Esquema general del sistema Shibboleth*

Para dar solución a este conjunto de requisitos, Shibboleth utiliza el concepto de SH (Shibboleth Handler), que no es más que un identificador asociado al usuario. Este manejador es utilizado por el SHAR para solicitar los atributos de un usuario. Cuando un elemento SHAR necesita realizar una consulta de atributos, Shibboleth le proporciona un SH con el que podrá preguntar identificar al usuario en cuestión. La Fig. 4-3 muestra una visión general de la arquitectura Shibboleth, donde se puede observar qué elementos intervienen y qué organización los aloja, si la origen del usuario o la propietaria del servicio.

4.3.2.2.1. Obtención del Shibboleth Handler

El primer paso que debe dar un elemento (por ejemplo el SHAR) que necesite el identificador SH del usuario, es contactar con su SHIRE (Shibboleth Handler Requester), tal y como muestra la Fig. 4-4. Este servicio es le responsable de obtener el identificador adecuado del usuario en cuestión. Para ello, el SHIRE contacta con el HS (Handler Service) de la organización origen del usuario, y le envía una consulta de tipo AQHR (Attribute Query Handle Request). El HS (Hanler Service) es responsable de proporcionar un identificador válido para dicho usuario.

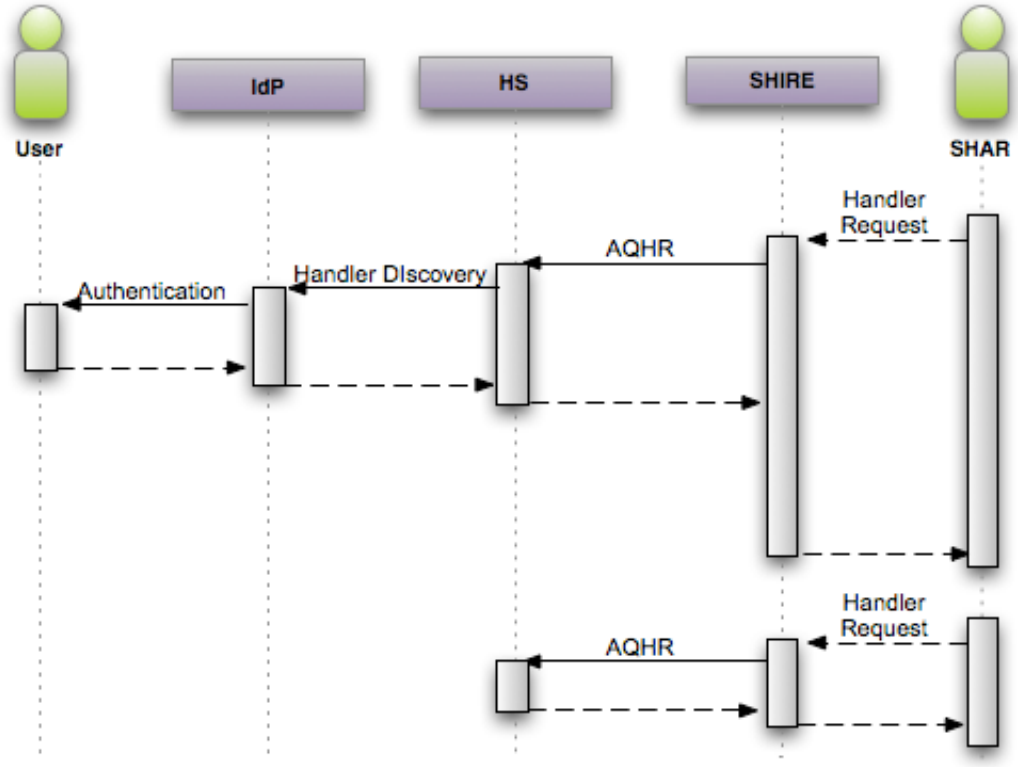

Fig. 4-4 *Diagrama de secuencia del la fase de obtención del Handler asociado al usuario*.

Si el HS no es capaz de identificar al usuario, éste tendrá que ser autenticado en un Identification Provider (IdP), asociado a dicho Hanlder Service. El IdP procederá a la autenticación del usuario mediante el método de autenticación configurado. Si la autenticación es satisfactoria, el HS proporcionará un identificador válido para este usuario al servicio SHIRE que lo requiera. En posteriores consultas al SH en relación a un usuario ya autenticado, el SH proporcionará el identificador del usuario sin requerir nuevas autenticaciones por parte de este, tal y como muestra la parte inferior del diagrama de secuencia de la figura 8.

4.3.2.3. Where Are You Service

Cuando se trabaja en entornos de múltiples organizaciones Shibboleth proporciona un servicio WAYF (Where Are You From). Este servicio permite conocer la organización origen de un usuario no identificado en la federación. Este servicio consigue realizar una relación entre un usuario de la federación y la organización origen a la que pertenece.

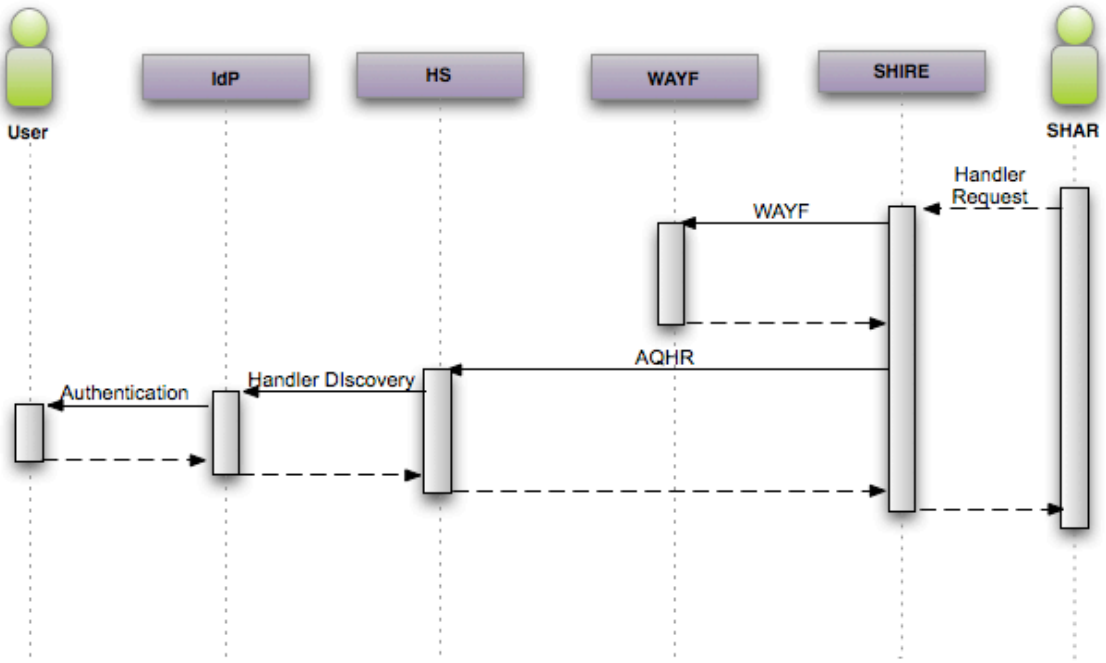

Fig. 4-5 *Diagrama de secuencia del uso del servicio WAYF*

Como se puede observar en el diagrama de secuencia de la Fig. 4-5, cuando un SHIRE es preguntado por la identidad de un usuario que está intentando acceder a un recurso, si éste no es capaz de identificar al dicho usuario, ni sabe cuál se su organización origen, el servicio SHIRE no sabrá a qué Handler Service preguntar. Para conocer éste dato, se realiza una consulta al servicio WAYF, y este devuelve al SHIRE el HS correspondiente a la organización origen del usuario a identificar.

4.4. Comparativa entre PAPI, Shibboleth y OpenID

Tanto Shibboleth, OpenID y PAPI son soluciones tecnológicas a tener en cuenta para dar respuesta al problema planteado al comienzo en este punto. Las tres arquitecturas:

- Mantienen un esquema de funcionamiento distribuido que permite diferenciar a nivel de infraestructura las organizaciones origen de los usuarios y las organizaciones propietarias del recurso. Por lo tanto son coherentes con un sistema multi-organziación.

- Son capaces de seguir un modelo de funcionamiento single sign on, con importantes diferencias en cuanto a los tipos de clientes soportados.
- Son aptas para usuarios móviles, no dependiendo en ningún casos de la dirección IP origen de las conexiones.
- Son compatibles con los principales navegadores: MS IExplorer, Mozilla, Netscape y Opera.
- Permiten el control de acceso a servidores web y servidores de aplicación basados en HTTP y HTTPS
- Analizadas globalmente, Shibboleth y PAPI mantienen muchos puntos en común, y en sus componentes se pueden observar como muchas de las funcionalidades que aportan una y otra solución se encuentran en ambos sistemas pero con nombres y empaquetados en subsistemas distintos. Por otro lado OpenID comparte mecanismos y soluciones con Shibboleth, pero difiere en mayor medida del modelo PAPI.

4.4.1. Identificador único

Todas las soluciones a analizar utilizan un sistema de identificación único de usuario, que permite dar sentido al concepto de federación. Gracias a este elemento se consigue, en un entorno multi-organización, por un lado identificar a un usuario en todos los componentes de la federación. Es decir, que los accesos que realiza un usuario en la federación quedan completamente diferenciados de los que realiza otro usuario.

En el caso de Shibboleth el identificador único, el handler asociado al usuario, es manejado por los elementos SHIRE (en la organización propietaria del recurso o servicio) y HS (en la organización origen del usuario). Todos los accesos que realice el usuario a los recursos asociados a un mismo SHIRE, serán identificados de forma única.

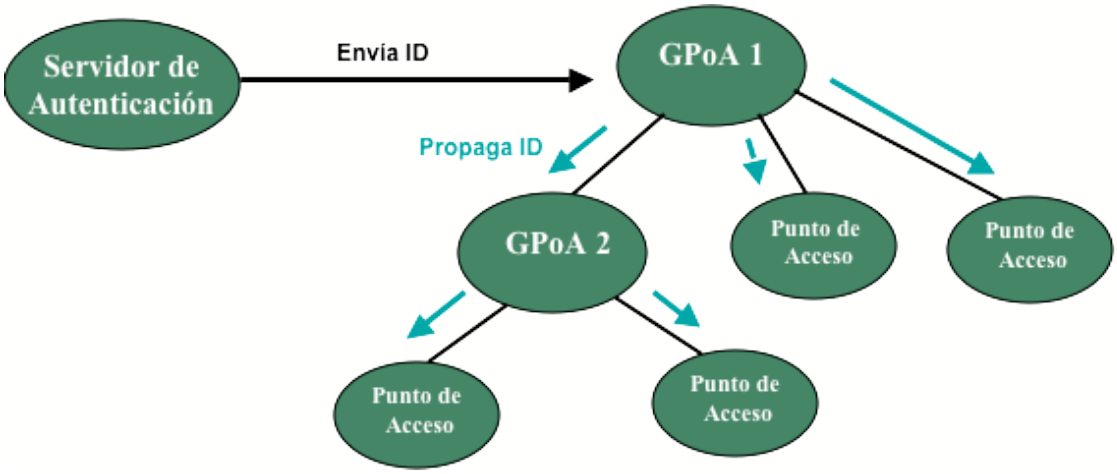

Fig. 4-6 *Modelo de identificador único en PAPI*

En el caso de PAPI, como muestra la Fig. 4-6, el identificador es generado por el Servidor de Autenticación de la organización origen y utilizado por los PoAs y GPoAs de las organizaciones destino (propietarias de los recursos). En este caso los GPoAs son capaces de propagar un identificador de usuario a PoAs o GPoAs por debajo jerárquicamente. Esta propagación no es síncrona, sino que se va produciendo en la medida que PoAs o GPoAs hijos solicitan a su GPoA padre la identificación del usuario.

En el caso de OpenID, el identificador se gestiona mediante el proveedor OpenID. Cualquier recurso que quiera conocer la identidad de un usuario, tendrá que contactar con el proveedor OpenID de dicho usuario.

Un aspecto muy importante respecto a la gestión de identidad del usuario es la capacidad que tiene la infraestructura para preservar, en los casos que así se precise, el anonimato del usuario, a la vez que se mantiene la capacidad de control y trazabilidad para el estudio de incidencias a nivel de seguridad. Tanto PAPI, como Shibboleth sí poseen la capacidad de anonimizar la identidad del usuario, incluso en el acceso a ciertos recursos. En ambos casos, al usuario se le asigna un identificador "aleatorio", válido para una sesión, y que sólo puede ser resuelto cruzando la información disponible en la organización del usuario. En el caso de OpenID, no se contempla actualmente esa posibilidad aunque sí podrían implantarse mecanismos para conseguirla.

Desde el punto de vista del usuario, el mecanismo de propagación de su identidad, debería ser completamente transparente. Por parte de Shibboleth esto no es posible, ya que al cambiar de dominio SHIRE, el usuario debe interactuar con el sistema para indicarle cuál es su organización origen. En el caso de PAPI, este problema está resuelto ya que la flexibilidad en la estructuración de PoAs y GPoAs hace fácil el poder crear dominios de identidad que engloben a todos los puntos de acceso y servidores de autenticación de la federación. En el caso de OpenID habría que forzar a que todos los usuarios de la federación utilizaran un proveedor de identidad común.

4.4.2. Definición de políticas de acceso

Uno de los objetivos de la federación es poder dotar de seguridad a servicios y recursos web, para que sean accesibles de forma segura por usuarios de la federación. Para conseguir este objetivo es imprescindible poder establecer reglas de control de acceso que fijen las condiciones que hacen un recurso accesible para un usuario. Se deben poder definir reglas de acceso en base a propiedades del usuario (organización, cargo, proyectos involucrados), y a condiciones de la consulta, tales como la hora a la que se realiza la consulta, o valores de ciertos parámetros incluidos en ella.

En este caso existe una gran diferencia entre PAPI o Shibboleth y OpenID, ya que este último no tiene definido ningún mecanismo de control de acceso.

En el caso de Shibboleth, el lenguaje de definición de reglas de control de acceso se limita prácticamente a poder definir criterios en base a qué atributos son necesarios y qué valores deben de estar contenidos en dicho atributos. Estos criterios pueden relacionarse mediante operaciones lógicas AND y OR.

En le caso de PAPI, y como ya se ha comentado en el punto X, posee un rico lenguaje para la definición de reglas de control de acceso. En él se incluyen: operaciones lógicas (AND, OR y NOT), funciones numéricas, funciones de cadenas de caracteres, funciones de fecha, etc. Además como argumentos a dichas funciones se pueden incluir parámetros de la consulta, atributos del usuario, o incluso, parámetros de entorno como la hora, la IP, etc.

En este punto tiene una clara ventaja PAPI frente al resto ya que es capaz de implementar reglas de control de acceso muy complejas y en función de muchos parámetros relativos al usuario y a la consulta.

4.4.3. Mantenimiento de sesión

Un elemento imprescindible en este tipo de infraestructura es el concepto de sesión, que normalmente va ligado al intervalo desde que el usuario se autentica "Login" hasta que el usuario finaliza la sesión a través de una operación "Loggout". La tecnología tipo web se basa en un esquema cliente - servidor sin estado, es decir, en un esquema consulta - respuesta en el que el servidor no está obligado a mantener un estado de ejecución de un cliente en concreto. Por otro lado, como ya se ha comentado, las infraestructuras de autenticación y autorización deben poder implementar el concepto de sesión ("Login" - "Logout").

Desde el punto de vista de la comparativa vuelven a surgir diferencias entre las tres soluciones. En este caso, OpenID no define ningún mecanismo de mantenimiento de sesión, mientras que PAPI y Shibboleth sí implementan mecanismos de mantenimiento de sesión mediante "cookies" cifradas. Este sistema se basa en el envío al usuario, incluido en las comunicaciones, "cookies" cifradas (que no pueden ser modificadas más que por el mecanismo de control de acceso que las creó). Con ello, el componente que realiza el control de acceso (PoA en PAPI y Resource Manager en Shibboleth) evita el tener que estar autenticando al usuario en cada petición.

Respecto a este punto, cabe destacar que PAPI implementa un complejo sistema de refresco y rotación de "cookies" que permite garantizar:

- La integridad de las "cookies" de sesión, es decir, que el usuario no ha modificado las "cookies" de sesión, al igual que Shibboleth.

- Que las "cookies" no se están utilizando desde otro equipo (por haberse copiado). Es decir, se detectan colisiones de dos equipos usando las mismas "cookies". Incluso aunque los dos equipos compartan la IP, mediante soluciones NAT o el uso de proxies.

- Que la rotación de "cookies" es consistente aunque durante la comunicación existan errores o que el usuario pare la carga de una página.

4.4.4. Protocolos

En este punto es donde se encuentra una de las principales diferencias entre PAPI y Shibboleth. PAPI mantiene un modelo tipo REST (Representational State Transfer),

basado en consultas HTTP tipo GET y con un protocolo sumamente ligero y a propósito de las necesidades encontradas a lo largo de estos años de utilización.

EL	Assertion	
ATT	AssertionID	
ATT	MajorVersion	
ATT	MinorVersion	
ATT	Issuer	MUST contain FQDN of the issuing HS
ATT	IssueInstant	
EL	AuthenticationStatement	MUST appear once and only once
ATT	AuthenticationMethod	See below
ATT	AuthenticationInstant	MUST equal time of origin site authentication
EL	Subject	See section 6.2 for subject information
EL	AuthenticationLocality	MAY be omitted
ATT	IPAddress	MUST equal client's dotted decimal IP address
EL	Conditions	
EL	AudienceRestrictionCondition	
EL	Audience	MUST equal SHIRE acceptance URL
EL	Advice	MAY be omitted
EL	shib:AttributeAuthority*	MAY appear zero or more times
ATT	Protocol	MUST identify a SHAR/AA exchange protocol
ATT	Binding	MUST point to AA via the chosen protocol
EL	ds:Signature	MUST contain an XML signature MAY include an X.509 certificate

Fig. 4-7 *Modelo de mensaje en respuesta a un Attribute Query Handle Request en Shibboleth*

Ejemplo de mensaje para paso de identificador y atributos ("assertion") entre SA y PoA:

"https://POA.URL.domain/PAPI/cookie_handler.cgi?ACTION=LOGIN&AS=test_AS&DATA=472364723647251352635172531265367153127531273512653126351263512351263512653126531265312653"

En este ejemplo se pueden identificar los campos:

ACTION, AS y DATA, que se han detallado en el apartado 3.5.3.

En el caso de Shibboleth, utiliza mensajes basados en SAML de OASIS. Este protocolo se basa en el XML firmado [47]. El mismo tipo de mensaje antes descrito codificado en SAML presentaría el formato que se muestra a continuación.

```
<samlp:Response
  xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
  IssueInstant="2003-04-17T00:46:02Z"
  MajorVersion="1" MinorVersion="1"
  Recipient="https://sp.example.com/Shibboleth.shire"
  ResponseID="_c7055387-af61-4fce-8b98-e2927324b306">
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#_c7055387-af61-4fce-8b98-e2927324b306">
        <ds:Transforms>
          <ds:Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            <InclusiveNamespaces PrefixList="#default saml samlp ds xsd xsi"
              xmlns="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:Transform>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>TCDVSuG6grhyHbzhQFWFzGrxIPE=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
      x/GyPbzmFEe85pGD3c1aXG4Vspb9V9jGCjwcRCKrtwPS6vdVNCcY5rHaFPYWkf+5
      EIYcPzx+pX1h43SmwviCqXRjRtMANWbHLhWAptaK1ywS7gFgsD01qjyen3CP+m3D
      w6vKhaqledl0BYyrIzb4KkHO4ahNyBVXbJwqv5pUaE4=
    </ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>
          MIICyjCCAjOgAwIBAgICAnUwDQYJKoZIhvcNAQEEBQAwgakxCzAJBgNVBAYTAlVT
          MRIwEAYDVQQIEwlXaXNjb25zaW4xEDAOBgNVBAcTB01hZGlzb24xIDAeBgNVBAoT
          F1VuaXZlcnNpdHkgb2YgV2lzY29uc2luMSswKQYDVQQLEyJEaXZpc2lvbiBvZiBJ
          bmZvcm1hdGlvbiBUZWNobm9sb2d5MSUwIwYDVQQDExxIRVBLSSBTZXJ2ZXIgQ0Eg
          LS0gMjAwMjA3MDFBMB4XDTAyMDcyNjA3Mjc1MVoXDTA2MDkwNDA3Mjc1MVowgYsx
          CzAJBgNVBAYTAlVTMREwDwYDVQQIEwhNaWNoaWdhbjESMBAGA1UEBxMJQW5uIEFy
          Ym9yMQ4wDAYDVQQKEwVVQ0FJRDEcMBoGA1UEAxMTc2hpYjEuaW50ZXJuZXQyLmVk
```

```
        dTEnMCUGCSqGSIb3DQEJARYYcm9vdEBzaGliMS5pbnRlcm5ldDIuZWR1MIGfMA0G
        CSqGSIb3DQEBAQUAA4GNADCBiQKBgQDZSAb2sxvhAXnXVIVTx8vuRay+x50z7GJj
        IHRYQgIv6IqaGG04eTcyVMhoekE0b45QgvBIaOAPSZBl13R6+KYiE7x4XAWIrCP+
        c2MZVeXeTgV3Yz+USLg2Y1on+Jh4HxwkPFmZBctyXiUr6DxF8rvoP9W7O27rhRjE
        pmqOIfGTWQIDAQABox0wGzAMBgNVHRMBAf8EAjAAMAsGA1UdDwQEAwIFoDANBgkq
        hkiG9w0BAQQFAAOBgQBfDqEW+OI3jqBQHIBzhujN/PizdN7s/z4D5d3pptWDJf2n
        qgi7lFV6MDkhmTvTqBtjmNk3No7v/dnP6Hr7wHxvCCRwubnmIfZ6QZAv2FU78pLX
        8I3bsbmRAUg4UP9hH6ABVq4KQKMknxu1xQxLhpR1ylGPdiowMNTrEG8cCx3w/w==
      </ds:X509Certificate>
    </ds:X509Data>
  </ds:KeyInfo>
</ds:Signature>
<samlp:Status><samlp:StatusCode Value="samlp:Success"/></samlp:Status>
<saml:Assertion
  xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
  AssertionID="_a75adf55-01d7-40cc-929f-dbd8372ebdfc"
  IssueInstant="2003-04-17T00:46:02Z"
  Issuer="https://idp.example.org/shibboleth">
  <saml:Conditions
    NotBefore="2003-04-17T00:46:02Z"
    NotOnOrAfter="2003-04-17T00:51:02Z">
    <saml:AudienceRestrictionCondition>
      <saml:Audience>http://sp.example.com/shibboleth</saml:Audience>
    </saml:AudienceRestrictionCondition>
  </saml:Conditions>
  <saml:AuthenticationStatement
    AuthenticationInstant="2003-04-17T00:46:00Z"
    AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
    <saml:Subject>
      <saml:NameIdentifier
        Format="urn:mace:shibboleth:1.0:nameIdentifier"
        NameQualifier="https://idp.example.org/shibboleth">
        3f7b3dcf-1674-4ecd-92c8-1544f346baf8
      </saml:NameIdentifier>
      <saml:SubjectConfirmation>
        <saml:ConfirmationMethod>
          urn:oasis:names:tc:SAML:1.0:cm:bearer
        </saml:ConfirmationMethod>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:SubjectLocality IPAddress="127.0.0.1"/>
  </saml:AuthenticationStatement>
</saml:Assertion>
 </samlp:Response>
```

Tal como indica el protocolo SAML, se trata de mensajes XML firmados mediante clave pública. En este mensaje se pueden observar tanto los campos definidos para el protocolo más aquellos correspondientes a la firma XML.

En el caso de OpenID, se utilizan formatos de mensaje propietarios, que se pueden dividir entre: el protocolo de asociación entre el propietario del recurso y el proveedor OpenID, y el resto de comunicaciones basadas en redirecciones de la aplicación cliente del usuario. El conjunto de comunicaciones basadas en las redirección del usuario, se realizan mediante redirecciones HTTP tipo GET (debido a la poca longitud del mensaje) y sus campos coinciden en gran medida con los utilizados por PAPI.

La ventaja de PAPI y OpenID en este punto, es que los protocolos son muy sencillos de implementar y muy poco pesados. Además los mecanismos de redirección basados en redirecciones HTTP de tipo GET son mucho más compatibles cara a integrar otro tipo de aplicaciones que no sean navegadores web. Por contra, Shibboleth implementa SAML que es un protocolo que contempla una gran casuística y es el auténtico estándar en sistemas de autenticación y autorización.

Por último conviene destacar el problema que presenta OpenID a nivel de protocolo en la fase de asociación entre el propietario del recurso que el proveedor de información. Gracias a esta fase, se elimina de tener que utilizar un mecanismo de confianza entre el proveedor de identidad y el propietario del recurso, como la distribución de las claves públicas en el caso de PAPI o la infraestructura de certificados de Shibboleth. El problema que tiene esta comunicación es que se realiza directamente sin utilizar la aplicación cliente del usuario como transmisor, como pasa en los mecanismos de redirección ya comentados. Esto hace que el acceso al proveedor OpenID deba esta abierto a todo Internet, y así poder recibir peticiones de asociación. Esto puede derivar en problemas con los firewalls intermedios y con las políticas de filtrado de las organizaciones.

4.4.5. Mecanismos de redirección

Los tres sistemas utilizan la redirección como mecanismo de comunicación entre elementos de la arquitectura. En concreto PAPI y OpenID utilizan la redirección basada en la respuesta 301, 302 de HTTP, obteniendo la URL a la cual el cliente debe ser redireccionado del campo Location de dicho mensaje de respuesta. Por lo tanto las

redirecciones en PAPI y OpenID son de tipo GET, independientemente cual haya sido el tipo de consulta inicial: GET, POST, PUT, etc.

En el caso de Shibboleth, y debido al tamaño de su protocolo SAML, el mecanismo de redirección empleado es mediante redirecciones HTML y javascript, como la que muestra la Fig. 4-8.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
     lang="en">
  <head>
    <title>You have been Shibbolized!</title>
  </head>
  <body onload="shib.submit()" style="text-align:center">
    <p>Ready to transmit your Shibboleth handle...</p>
    <form name="shib" action="https://frobozzica.com/shire"
        method="POST">
      <input type="hidden" name="TARGET"
          value="https://frobizzica.com/restricted/"/>
      <input type="hidden" name="SAMLAssertion"
          value=" bHMgZGlyIC9oIC93ICUmDQ...9wIlwNCg==" />
      <input type="submit" value="Transmit"/>
    </form>
  </body>
</html>
```

Fig. 4-8 *Mecanismo de redirección utilizado por Shibboleth*

Este tipo de redirección no conlleva más problemas en un entorno de navegador web, pero a la hora de integrar otro tipo de aplicaciones, como aquellas basadas en librerías HTTP, puede tener complicaciones ya que no hay librerías HTTP estándar que implementen este mecanismo de redirección.

4.4.6. Estructura de confianza

En PAPI la confianza entre elementos del sistema, se implementa gracias a cifrado y firma basados en clave pública (PKI), así como la estructura jerárquica formada entre PoAs y GPoAs. De esta forma un elemento que se identifica ante un PoA, si no es reconocido, se le redirige a su GPoA y así hasta que uno de ellos lo identifica. A continuación, es vuelto a redirigir hasta llegar al PoA original, con información relacionada con el elemento a identificar. Todos estos mensajes, son firmados para conseguir la autenticación del emisor y la preservar la integridad del mensaje. Entre

Servidores de autenticación y el GPoA de la federación, hay que mantener un intercambio de clave pública.

En el caso de Shibboleth, es necesario montar una Jerarquía de certificación. Es decir hay que montar una autoridad de certificación asociada a la federación, ya que en Shibboleth no existe el concepto de jerarquía entre elementos (por ejemplo entre SHIREs), por lo que un elemento de cualquier nivel debe poder comunicarse con otro (por ejemplo un SHIRE de un servidor, con el HS de un laboratorio), por lo que hay que distribuir certificados firmados por la CA de la federación entre todos los elementos de Shibboleth.

El mantener una Autoridad de Certificación (CA) asociada a la federación, en un principio, y salvo que hubiera otro tipo de beneficio colateral, no resulta beneficioso, ya que introduce mucha carga de gestión. Una CA de estas características, implicaría mantener una autoridad de registro además de la CA y un sistema de revocación apropiado.

En el caso de OpenID no se define una estructura de confianza, por lo que todo se basa en mantener una lista de OpenIDs en los que creer.

4.4.7. Clientes soportados

En el caso de PAPI, están soportados los navegadores web más extendidos, como: MS IExplorer, Mozilla, Netscape y Opera. Así mismo son compatibles aplicaciones JAVA que utilicen conexiones HTTP o túneles HTTP implementados mediante:

Librería HTTP nativa: para JAVA superior a la versión 1.5. Ya que es a partir de esta versión cuando la librería "java.net" contempla el uso de manejadores de "cookies".

Librería commons-httpclient: englobada en el proyecto Jakarta de Apache y utilizada como base para protocolos como XML-RPC, que quedan automáticamente soportados por PAPI.

En la comunidad PAPI, se han conseguido adaptadores con el objetivo de utilizar la autenticación PAPI en entorno Citrix.

En el caso de Shibboleth y OpenID, el sistema de single sign on, funciona para recursos accesibles mediante los navegadores web antes mencionados.

Este es un punto realmente importante, ya que atañe a uno de los requisitos impuestos para la federación. El protocolo y estructura organizativa utilizado por PAPI le permite mayor flexibilidad cara a soportar más aplicaciones y protocolos.

4.4.8. Manejo de atributos

En el caso de PAPI, posee un sistema de consulta de atributos basado en una petición tipo "ATTREQ", que es redireccionada por toda la jerarquía hasta llegar al servidor de autenticación, que una vez evaluada la petición, envía la respuesta, también utilizando mecanismos de redirección, hasta el PoA que generó la consulta. Este sistema de petición-respuesta de atributos, tiene limitaciones en cuanto al tamaño de la respuesta, ya que las redirecciones se resuelven mediante consultas tipo GET, lo que introduce una limitación de 4096 Bytes. Aún así para la mayoría de los casos, es un tamaño de mensaje más que suficiente. PAPI en su concepción, está diseñado como un sistema PUSH, donde los atributos necesarios para un recurso son enviados del servidor de autenticación al PoA en el último paso de la autenticación. O bien, los atributos son primero enviados al GPoA, apara que éste los distribuya posteriormente a sus PoAs hijos. En cualquier caso, PAPI maneja los atributos distribuyéndolos antes de ser demandados.

En el caso de Shibboleth, es un sistema tipo PULL, es decir, los atributos son requeridos por los SHAR a las attribute authority, en el momento que son necesarios. De hecho el modelo está mucho más orientado hasta intercambio, por lo que se estructura toda una CA que permite la firma de mensajes entre cualesquiera elementos del sistema y por tanto la comunicación directa entre ellos.

En el caso de OpenID, se ha definido un esquema de gestión de atributos que incluye la consulta y respuesta entre el propietario del recurso y el proveedor OpenID, además de mecanismos para que el usuario pueda almacenar sus atributos en el proveedor OpenID.

A pesar de que tanto PAPI como OpenID, consiguen implementar mecanismos de distribución de atributos válidos, Shibboleth implementa un sistema de manejo de atributos más desarrollado a todos los niveles: definición de procedimientos, protocolo de intercambio, política de distribución de atributos, etc.

4.4.9. Servidores y recursos soportados

En el caso de PAPI, existen implementaciones para servidores Apache, Tomcat y páginas PHP. Además existe un elemento "PAPI proxy" que permite redireccionar consultas HTTP a servidores y servicios que se encentran alojados en tos servidores. Con ello se consigue dar servicio a prácticamente cualquier tipo de servidor HTTP que maneje tecnología HTML en sus respuestas: Apache, IIS, Tomcat, Web Logic, etc.

Shibboleth posee diferentes implementaciones para plataformas como "Apache" y "Microsoft IIS". Además se han implementado módulos de acceso para diversos sistemas web: JSTORE, Moodle, Drupal, etc.

OpenID posee implementaciones en muy diferentes lenguajes de programación y hay disponibles módulos para la plataforma Apache y diferentes sistemas web: WordPress, Drupal, Movable Type, etc.

En este punto existe bastante equilibrio entre todos los sistemas. Quizá quepa destacar la posibilidad que da PAPI para controlar el acceso a servidores remotos sin necesidad de instalar nada en ellos, gracias a la tecnología de "proxy" inverso que incluye.

4.4.10. Conclusión de la evaluación

Finalmente se opta por PAPI como sistema para implementar la federación, por los siguientes motivos:

- Facilidad de instalación y gestión: PAPI, gracias a su "PAPI proxy", puede ser instalado como un appliance en una organización con un mínimo soporte. No requiere tocar los servicios a controlar, ya que toda la configuración se realiza en el "proxy".
- Soporta aplicaciones JAVA: Este es uno de los requisitos que se detallaban en las funcionalidades que debía soportar la federación, debido al número de aplicaciones JAVA desarrolladas para el uso de los científicos.
- Su esquema de redireccionamiento es compatible con librerías HTTP estándar, no requiriendo soluciones javascript como en Shibboleth.

- Estructura de confianza sencilla y fácil de gestionar: PAPI no requiere de implantar una CA como en el caso de Shibboleth. Y por otro lado permite la definición de estructuras organizativas en base a GPoAs, cosa que en el casod e OpenID obliga a la implementación de un proveedor OpenID único.
- Permite implementar reglas complejas de control de acceso a los recursos.
- Implementa un completo mecanismo de mantenimiento de sesión.

4.5. La federación basada en PAPI

Como se comenta en el punto anterior, se elige PAPI como IAA (Infraestructura de Autenticación y Autorización) para soportar la federación EFDA [41,48]. Esta decisión no es irreversible, ya que por la naturaleza tan similar entre todas las soluciones que se manejan hoy en día como IAA, se podría, aun coste aceptable, cambiar de IAA manteniendo todos los demás elementos y funcionalidades de la federación.

4.5.1. Estructura federativa PAPI

En una federación la cantidad de recursos en los que un usuario debe estar identificado puede llegar a ser muy alta. Realizando una estimación a groso modo, suponiendo 10 organizaciones federadas y que cada una de ellas comparta unos 15 recursos entre zonas web y servicios, un usuario podría necesitar identificarse en 150 PoAs. Para paliar este problema, ya comentado previamente, se utilizan componentes GPoA. Estos elementos organizados de forma jerárquica permiten expandir la identidad de un usuario desde un nodo a sus nodos hijos. Por lo que si un usuario está identificado correctamente y posee las credenciales ("cookies" PAPI) de un GPoA, estará asimismo identificado cuando intente acceder a cualquiera de los PoAs o GPoAs que se encuentren por debajo en la jerarquía.

En función de lo dicho anteriormente, a la hora de montar una federación basada en PAPI, existen diversas alternativas, aunque son dos las más razonables. La primera se fundamenta en disponer de un GPoA para servicios federados y un servidor de autenticación en cada organización, y registrar en cada SA de cada organización, los GPoAs de las organizaciones federadas, tal y como muestra la Fig. 4-9. Cuando un

usuario se autentica en el servidor de autenticación de su organización, adquiere credenciales de cada uno de los GPoAs de las organizaciones federadas, y por tanto, está correctamente identificado en todos los recursos que cuelguen de dichos GPoAs.

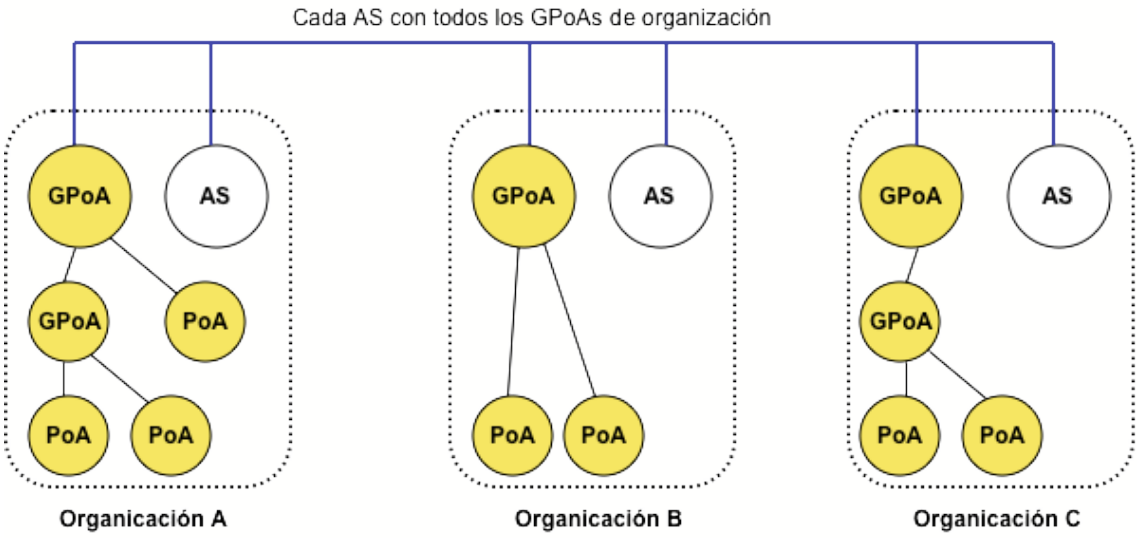

Fig. 4-9 *Estructura federativa de PAPI sin GPoA único*

A nivel de gestión, cuando se incluye una nueva organización en la federación, su GPoA debe ser registrado en cada uno de los SAs del resto de organizaciones. Aún así, si una organización quisiera incorporar un nuevo servicio a la federación, sólo tendría que registrar dicho servicio en el GPoA apropiado de su organización, el resto no tendría que realizar ninguna acción nada, ni tendrían que ser informados de nada.

Para la segunda opción, como se muestra en la Fig. 4-10, cada organización dispone (al igual que pasa con la anterior) de un GPoA para servicios federados y un servidor de autenticación. La diferencia está en que, en este esquema, los GPoAs de organización de las diferentes organizaciones se configuran como GPoAs hijos de un GPoA común a la federación que los engloba. En este tipo de estructura organizativa, cuando un usuario se autentica en el servidor de autenticación de su organización, obtiene una sola credencial, la credencial del GPoA de la federación, y con ella será correctamente identificado en cualquiera de los recursos o servicios federados. A nivel de gestión, el proceso se simplifica ya que la incorporación de una nueva organización a la federación, sólo representa un cambio en la configuración del GPoA de la federación. El resto de organizaciones no tendrían que realizar ningún cambio, ni tendrían que ser

informadas de los cambios. Por supuesto, a nivel de servicio, la inclusión de un nuevo recurso, no requiere más que ser correctamente configurado en el GPoA apropiado.

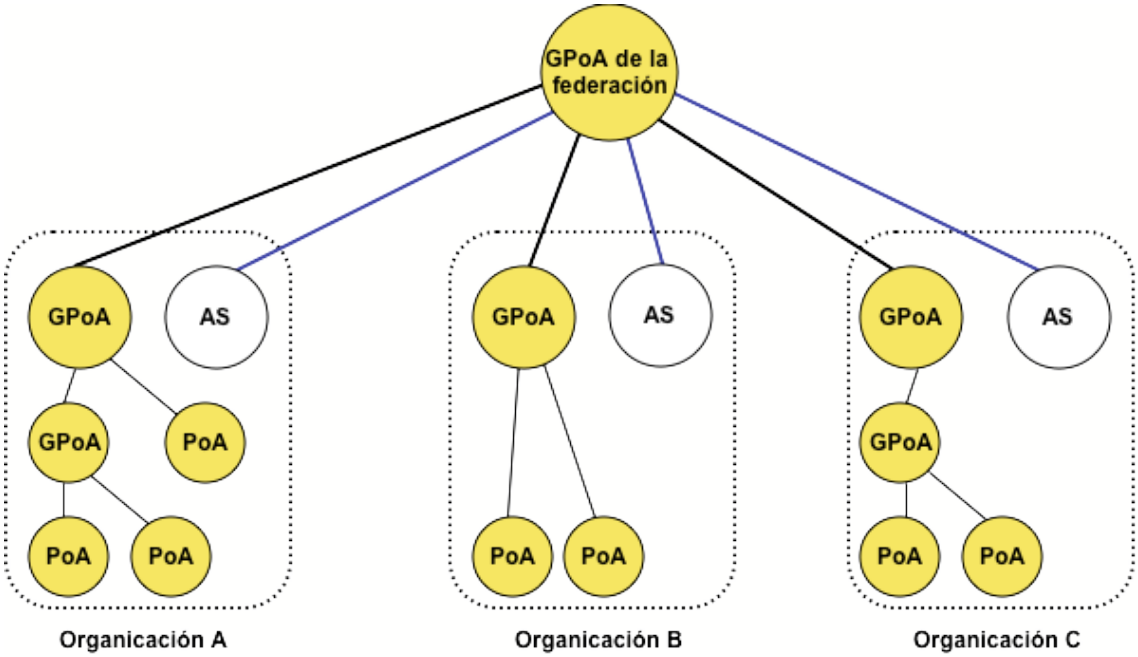

Fig. 4-10 *Estructura federativa de PAPI con GPoA único*

En una comparación de los dos modelos, el primero tiene la ventaja de que no incorpora un punto de fallo único al sistema, como ocurre con el segundo con el GPoA de federación. Así mismo, el tráfico de mensajes es más compensado al no pasar todos por el GPoA de la federación. Por contra, el segundo modelo es mucho más sencillo de gestionar, ya que en el primero: la configuración de cada organización es más compleja, cuando una nueva organización se incorpora a la federación el resto debe realizar cambios en su configuración, y por tanto también deben realizar pruebas, que implican a varias organizaciones, de que los cambios son correctos. Por contra la inclusión de una organización en el segundo modelo implica una sencilla comunicación y prueba con el gestor del GPoA de la federación. Por otro lado, cara al proceso de autenticación, el disponer de un sólo GPoA para toda la federación simplifica considerablemente el proceso de precarga de "cookies".

Los limitaciones en recursos humanos en las organizaciones implicadas, hacen que en proyectos que engloban a dichas organizaciones, la simplificación de procesos a nivel de gestión tenga una prioridad máxima. Por este motivo, y por que PAPI admite

configuraciones en alta disponibilidad (como prueba la instalación PAPI para acceso a publicaciones electrónicas implantada en la UNED (Universidad Nacional de Educación a Distancia), y debido a que el nivel de carga a nivel de tráfico que generan las redirecciones en el GPoA raíz es muy inferior a la capacidad de una red actual, finalmente se optó por la segunda alternativa, que aprovecha al máximo el potencial de las estructuras jerárquicas GPoA.

4.5.1.1. Where Are You From (WAYF) de PAPI

Aparte del sistema de propagación de identidad de PAPI basado en la agrupación de PoAs (los GPoAs), existe otra función que resulta imprescindible en un sistema federado de estas características, en el que la infraestructura de autenticación y autorización se basa en una solución distribuida y en el que existen varios servidores de autenticación integrados dentro de la federación. Se trata de un sistema que permite a los usuarios que todavía no han iniciado la sesión dentro de la federación, y por tanto son desconocidos para ella, seleccionar cuál es el servidor (de todos los posibles) que es válido para realizar el proceso de autenticación. Normalmente, este servidor coincide con el servidor de autenticación de la organización a la que pertenece el usuario. Esta función está soportada por el servicio WAYF (Where Are You From) de la federación.

Para la implementación del servicio WAYF, PAPI ofrece dos alternativas. Por un lado PAPI implementa internamente un servicio WAYF por defecto, que a partir de los servidores de autenticación registrados en el fichero de configuración, crea un mecanismo para que el usuario pueda elegir su servidor de autenticación de entre los posibles y posteriormente es redirigido a dicho servidor. La otra alternativa es que PAPI utilice un sistema externo para realizar esta función. En este caso el usuario es redirigido a una URL donde se localiza el servicio WAYF. Esta alternativa permite tener mucho más control sobre el esquema de interacción con el usuario, las opciones mostradas y la forma de hacerlo, así como la posibilidad de poder incluir tecnología que ayude a tomar la decisión. En el caso de la federación EFDA, el servicio WAYF se ha implementado mediante un CGI (Common Gateway Interface) implementado en Perl y que internamente mantiene la lista de servidores de autenticación integrados en la federación.

La secuencia y el formato de los mensajes intercambiados en todo el proceso, son descritos en detalle en el punto "Funcionamiento de la Federación".

4.5.2. Configuración a nivel de organización

Para facilitar la instalación de PAPI en las diferentes organizaciones que participaran en la federación se ha optado por fijar una configuración base inicial formada por un servidor PAPI que incluye:

- Un servidor de autenticación conectado al repositorio de usuarios de la organización.
- Un GPoA a nivel de organización para recursos federados. Este GPoA representará a la organización cara a la federación y por tanto tiene que tener configurado el GPoA de la federación como GPoA padre al que redirigir a los usuarios no identificados.
- PoAs configurados como hijos del PoA de la organización y que controlan el acceso a recursos de la organización. La particularidad de estos PoAs es que el recurso al que protegen puede estar localizado en otros servidores y el servidor PAPI utiliza la tecnología PAPI "proxy" para actuar como interfaz de acceso a los servicios asociados a los recursos a proteger.

Con este formato tipo "todo incluido", como se muestra en la Fig. 4-11, se ha creado una distribución "inicial" que tras su instalación, la organización dispone de un servidor completamente operativo y compatible con la federación EFDA, y que puede ser modificado con el tiempo para ajustarlo a la imagen corporativa y a las necesidades de control de acceso de la organización.

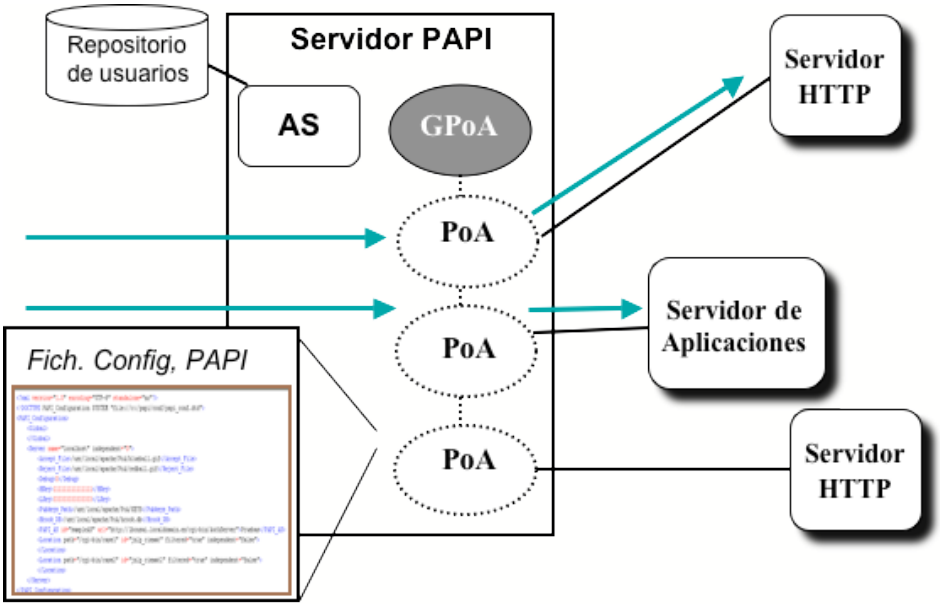

Fig. 4-11 *Arquitectura de un servidor PAPI en la federación EFDA*

Al disponer PAPI de tecnología de "proxy" inverso, el servidor PAPI actúa como interfaz de acceso de servicios a los que protege, pero que en muchos casos no necesitan ser adaptados. De esta forma PAPI tiene un modo de funcionamiento tipo cortafuegos a nivel de aplicación y en un formato que hace muy fácil su gestión en una primera fase, ya que todas las políticas de acceso (incluso de servicios remotos) se centralizan sobre un único fichero de configuración, y todos los componentes y tecnologías PAPI se concentran en una sola instalación.

Gracias al nuevo esquema de configuración de PAPI, se ha conseguido desacoplar todo lo relacionado con PAPI y la definición de políticas de acceso, respecto a Apache y sus directivas de configuración. Todo esto a repercutido en facilitar de forma considerable la gestión y focalizar los esfuerzos del técnico responsable en aspectos relacionados con PAPI y los servicios a proteger.

Como ya se ha comentado anteriormente, cara a facilitar la instalación de PAPI en las diferentes organizaciones se ha desarrollado un instalador capaz de dejar funcionando una configuración tipo "inicial" lista para ser enlazada en la estructura de la federación y con ejemplos de los principales configuraciones de puntos de acceso que se pueden dar:

- PoA local federado: Configuración básica en la que se protege una zona web local al servidor Apache donde funciona PAPI.

- PoA de acceso a recurso remoto: Configuración en la que se protege un servidor web remoto.
- PoA de acceso a CGI: En este ejemplo se muestra las reglas de control de acceso basada en atributos de la consulta, así como ejemplo del paso de los atributos del usuario a la aplicación protegida.
- PoA con acceso desde una aplicación JAVA: Este ejemplo muestra la protección de un CGI que es accedido mediante una aplicación JAVA. En este caso se muestra los procesos de carga de credenciales JAVA y la seguridad orientad a la conexión y no la aplicación cliente. Es decir, que si se realiza logout, la aplicación no se cierra pero sus consultas al servidor dejan de funcionar.

4.6. Funcionamiento de la federación

Un usuario dispone de varias procedimientos para acceder a recursos integrados en la federación en función del orden en que accedan al recurso y se autentiquen. En la federación se contemplan tanto los casos en los que el usuario intenta acceder aun recurso sin estar autenticado, como los casos en los que el usuario primero se autentica y luego accede a los recursos.

A continuación se va a describir las principales secuencias que se dan en la federación junto con el formato de cada uno de los mensajes que intervienen.

4.6.1. Usuario accede a recurso sin haberse autenticado

Esta secuencia se da en la federación cuando un usuario accede a un recurso sin haber iniciado la sesión en la federación, (Login), es decir, sin haberse autenticado previamente. Ante este tipo de casos, un sistema de autenticación tiene dos alternativas. La primera consiste en advertir al usuario de que no es posible identificarlo y que por tanto debe realizar un proceso de identificación, pero sin facilitarle más soluciones. La segunda alternativa consiste en, una vez el sistema detecta el problema de identificación, éste provee al usuario de una serie de mecanismos para solucionar el problema. PAPI y más concretamente la federación EFDA, opta por esta segunda vía, debido a la considerable mejora, tanto a nivel de usabilidad de la solución, como a nivel de la percepción del sistema por parte del usuario. Más concretamente, la

federación EFDA (gracias a PAPI) reconduce al usuario a través de los diferentes pasos que (si todo es correcto) le llevan finalmente a acceder al recurso solicitado.

El proceso previamente comentado, queda claramente descrito en el diagrama de la Fig. 4-12, en él la secuencia de acciones son:

1. "HTTP Request": El usuario inicia la consulta al recurso HTTP.
2. "Redirect GPoA Check" y "GPoA Check": Respuesta en la que se delega la identificación del usuario al GpoA de la federación.
3. Si el usuario ha no ha iniciado sesión dentro de la federación y por tanto no puede ser identificado:
    a. "Redir. WAYF Check" y "WAYF Check": Redirigir al usuario al servico WAYF para que seleccione el servidor de autenticación a utilizar.
    b. "Redir. AS ATTREQ" y "AS ATTREQ": Redirección al SA seleccionado para autenticar al usaurio. La consulta es de tipo ATTREQ porque es equivalente a una consulta de autenticación con petición de atributos, ya que el objetivo principal de ambas es identificar al usuario, y ambas tienen un URL de vuelta al que redirigir el usuario.
    c. "Redir. Checked", "Resp. Checked": Redirigir respuesta tipo "CHECKED" al GPoA de la federación.
4. "Redir. Checked", "Resp. Checked": El GPoA de la federación devuelve al PoA una respuesta tipo "CHECKED".
5. "HTTP Request": la consulta originaria HTTP se retoma y se resuelve contra el recurso protegido.
6. "HTTP Response": El resultado de la consulta HTTP inicial es devuelto al usuario.

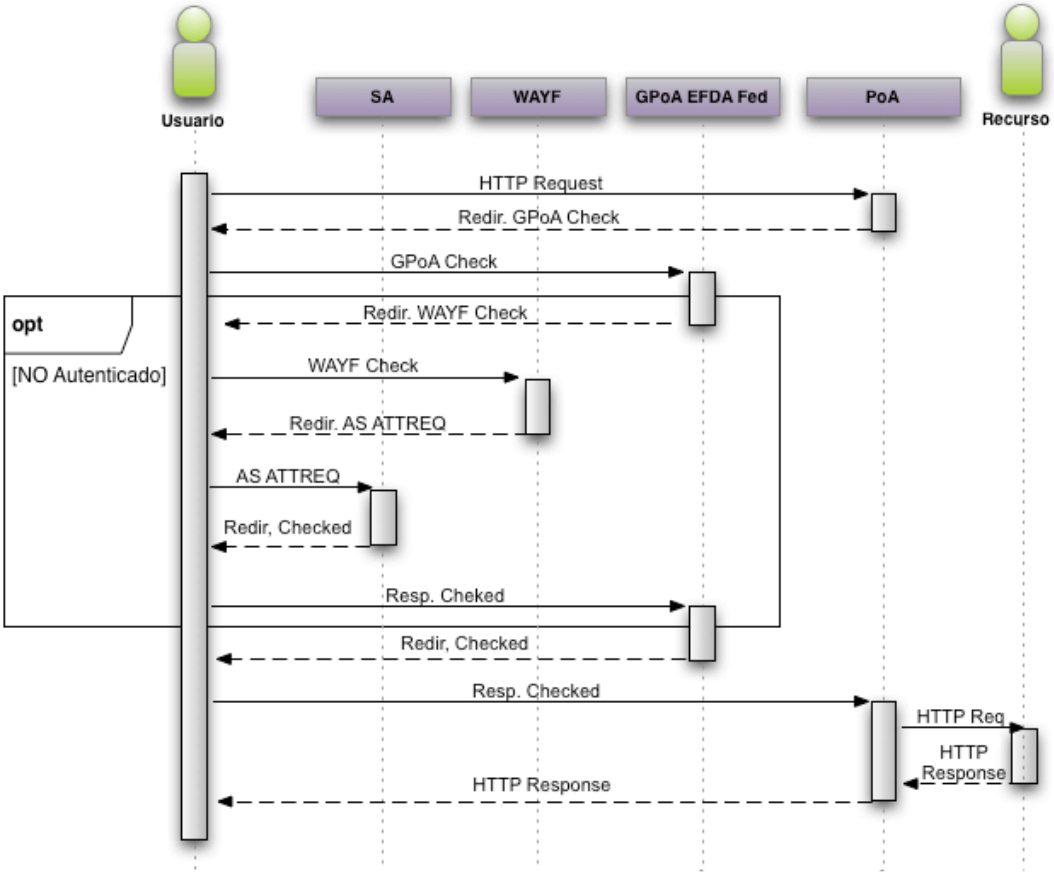

Fig. 4-12 *Diagrama de secuencia del protocolo PAPI en la federación EFDA*

4.6.2. Usuario primero se autentica y posteriormente accede al recurso

Este caso corresponde a un típico caso de sistemas en los que al usuario se le muestran los recursos a los que puede acceder (junto con las URLs de acceso), una vez éste a ha sido correctamente autenticado. Este caso, es más habitual en el entorno de una organización en el que existe una gestión local de todos los recursos disponibles. Sin embargo en un entorno tan extenso como el de la federación EFDA, en el que se pueden crear nuevos recursos que pueden implicar a un grupo de usuarios de diferentes organizaciones, este sistema no resulta fácil de implantar. Aún así, es perfectamente válido y como muestra la Fig. 4-13 su secuencia de pasos es:

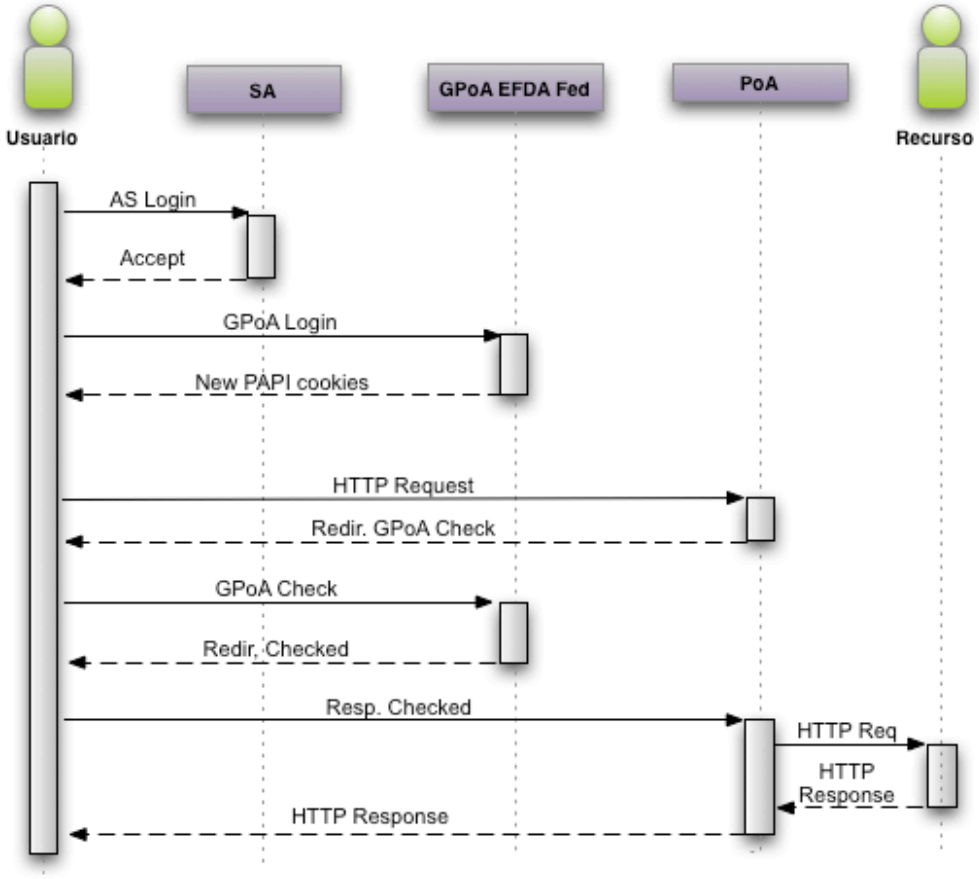

Fig. 4-13 *Diagrama de secuencia del protocolo PAPI en la federación EFDA de un usuario previamente autenticado*.

1. "AS Login": El usuario se autentica en el servidor de autenticación que le corresponde. En función del método de autenticación, así serán las credenciales que envía el usuario al servidor de autenticación.
2. "Accept": El servidor de autenticación (si todo ha sido correcto), envía una respuesta de "Accept" al usurio. Esta respuesta puede ser configurada mediante las plantillas que posee el servidor de autenticación de PAPI, pero en general, en esa respuesta se envía una lista de URLs de tipo "Login" firmadas, cada una de las cuales corresponde a un PoA o GPoA del cual se van a precargar "cookies" PAPI.
3. "GPoA Login": Por cada URL firmada en el mensaje "Accept ", enviado por el servidor de autenticación al usuario, el navegador web intenta resolver dicha URL, y envía unmensaje de tipo "GPoA Login", en este caso al GPoA de la federación.

4. "New PAPI cookies": el GPoA de la federación responde al "Login" enviado por el usaurio, con un mensaje de respuesta HTTP con cabereas "Set-Cookie" para las "cookies" PAPI correspondientes a dicho GPoA.
5. "HTTP Request": El usuario inicia la consulta al recurso HTTP.
6. "Redirect GPoA Check" y "GPoA Check": Respuesta en la que se delega la identificación del usuario al GpoA de la federación.
7. "HTTP Request": la consulta originaria HTTP se retoma y se resuelve contra el recurso protegido.
8. "HTTP Response": El resultado de la consulta HTTP inicial es devuelto al usuario.

Tanto en este caso, como en el visto en el punto anterior, subsiguientes accesos a cualquier recurso federado, sólo implica una petición de identificación del PoA correspondiente hacia el GpoA padre. Por simplificación, en el diagrama se corresponde con el GPoA de la federación. Por lo tanto la diagrama de secuencia se corresponde con el que muestra la Fig. 4-14.

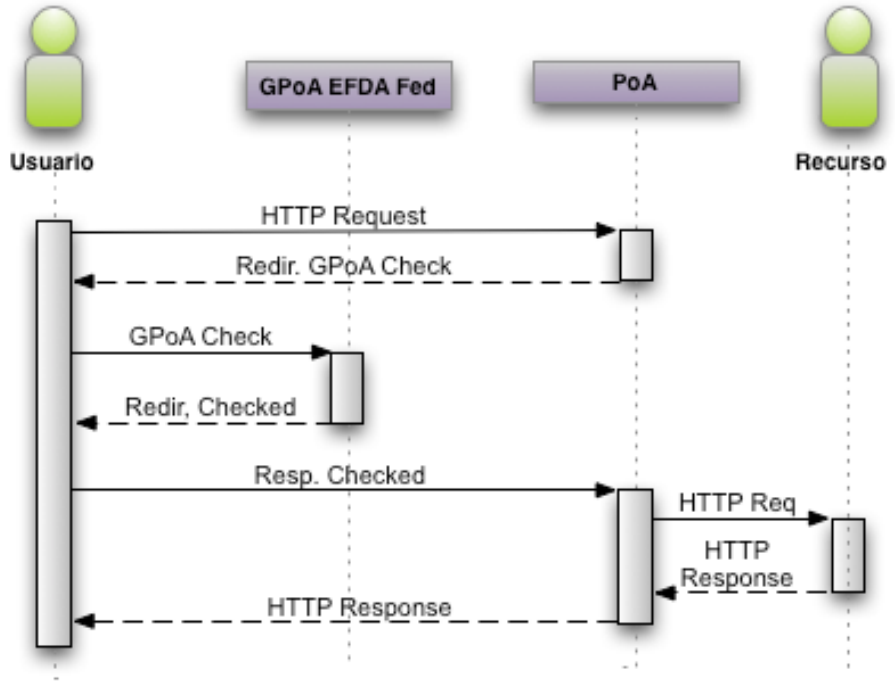

Fig. 4-14 *Diagrama de secuencia correspondiente al protocolo entre PoA y GPoA de la federación EFDA*

4.6.3. Mecanismo de "logout" real

Las federaciones que han sido implementadas utilizando infraestructuras distribuidas de seguridad, uno de los problemas que se les plantea es la implementación de un sistema de "logout" a nivel de toda la federación. Este problema no se da en sistemas basados en soluciones centralizadas, ya que en estos sistemas, existe un núcleo (servicio) central que autentica y autoriza y por tanto mantiene las sesiones de todos los usuarios. Sin embargo, en sistemas distribuidos, el software cliente mantiene claves de sesión (en el caso de PAPI son "cookies" cifradas) de diferentes servicios. Al hacer "logout", el servidor de autenticación, en el mejor de los casos, tiene registrados los servicios a primer nivel para los que el cliente mantiene sesiones activas, y de esos servicios es posible que pueda cerrar la sesión pero no mucho más allá. Para conseguir un "logout" total en este tipo de sistemas, habría que, o crear un proceso de difusión por todos los posibles recursos en los que el cliente pudiera tener una sesión abierta solicitando el cierre de la misma, o conseguir acceder al software cliente y generar un mecanismo que anulara las claves de sesión correspondientes a la federación.

En el caso del sistema PAPI, y por consiguiente en la federación EFDA, se ha conseguido implantar un sistema de cierre de sesión que funciona, tanto para los recursos accesibles desde el navegador web, como para los accesibles a través de las aplicaciones JAVA, aunque el proceso de "logout" es diferente en ambos casos.

El proceso de "Logout" sigue los siguientes pasos:

1- El usaurio accede a su servidor de autenticación correspondiente utilizando su navegador web, normalmente ya tiene una página web abierta en dicho servidor, y solicita el "Logout", haceindo clic en el botón o link correspondiente.

2- El servidor de autenticación responde al navegador web, si todo es correcto, enviándole una lista de URLs, equivalente a la que devuelve en caso de "logging", pero en este caso las URLs incluyen la orden LOGOUT.

3- Para cada URL el navegador web realiza la consulta y, por tanto, comunica la orden de LOGOUT a cada uno de los PoAs o GPoAs registrados.

4- Cada uno de los PoAs o GPoAs que reciben la orden de LOGOUT, le envían al navegador web un mensaje de respuesta HTTP que incluye, mediante cabeceras "Set-Cookie" la expiración de las "cookies" de sesión correspondientes.

Mediante este proceso ya se ha garantizado el borrado de credenciales a primer nivel. En el caso de la federación EFDA, se elimina la credencial correspondiente al GPoA de la federación.

A partir de aquí, una opción sería (como ya se ha comentado previamente) distribuir la orden de LOGOUT a todos los PoAs y GPoAs hijos en la jerarquía y así sucesivamente. En el caso de PAPI la agrupación de PoAs se hace configurando un GPoA al PoA a agrupar, pero no al revés, es decir, un GPoA no mantiene registro de los PoAs hijos. Esto se hace, por un lado por mantener en lo posible un esquema lo más distribuido posible, y como consecuencia, para facilitar la gestión de los recursos, ya que de esta forma, se pueden proteger recursos sin tener que estar informado al gestor del GPoA de la organización, ni al de la federación. Por este motivo se hace imposible propagar las acciones desde los GPoAs padre a los hijos.

Aún así, y aplicando el mismo esquema de funcionamiento que para el caso del "Login", se puede "forzar", con cierto grado de periodicidad, a que los PoAs consulten sobre la validez del usuario a sus GPoAs padre. De esta forma se consigue que si un GPoA ha realizado Logout, en un momento dado sus hijos se enteren de este hecho, y así puedan también hacer lo mismo. El mecanismo con el que los PoAs hijos realizan consultas a sus GPoAs padre vuelve a ser la respuesta de redirección "GPoA Check" que fuerza al navegador web del usuario a realizar una consulta al GPoA padre.

La clave en este esquema está en determinar la periodicidad con la que un PoA hijo vuelve a consultar con su GPoA padre, ya que la redirección antes descrita consume recursos de la infraestructura e incluye cierto retardo en la consulta. En el caso de PAPI, si el tiempo entre consultas es inferior a un umbral, que es configurable, entonces se supone que el usuario se encuentra activamente en ese servicio y no se fuerza a la consulta del GPoA. Sin embargo, en cuanto el tiempo entre consultas exceda de dicho umbral, se considera que el usuario ha podido realizar un logout durante ese periodo de tiempo, y se fuerza a la confirmación del GPoA padre. Si el GPoA padre, por ejemplo el de la organización, ha sido consultado recientemente desde otro PoA hijo (el usuario ha seguido activo pero en otra tarea), la consulta se resuelve y el resultado se devuelve al PoA original. En caso contrario, el GPoA seguirá suponiendo que el usuario ha podido hacer logout y reescalará la consulta a su GPoA padre (en este ejemplo corresponderá al de la federación). Los PoAs y GPoAs que reciben las ordenes tanto de LOGIN como de LOGOUT desde el servidor de autenticación, no reescalan las consultas a nadie ya que

ellos sí saben a ciencia cierta si el usuario ha realizado un cierre de sesión. Gracias a este mecanismo se consigue propagar el logout a todos los niveles de la federación, manteniendo el carácter distribuido del sistema PAPI, y penalizando de forma mínima el tiempo de respuesta en una sesión web normal.

En el caso de la federación EFDA, además se incluyen aplicaciones que hacen uso del repositorio común de "cookies". Para extender el cierre de sesión a dichas aplicaciones, se ha implementado una solución basada en la tecnología de "Loaders" de PAPI (descrita en detalle en el punto 3.8.2) . Este sistema se basa en la tecnología JWS y permite lanzar de forma segura un "Loader", que de forma transparente para el usuario, realiza la acción de carga de credenciales en el repositorio único de "cookies" para aplicaciones, y en el caso que nos ocupa, el borrado de dichas credenciales. Los primeros pasos de la secuencia son equivalentes que en el mecanismo de logout antes comentado, ya que ambos mecanismos se inician con la misma acción "Single-Sign-Off". La secuencia es la siguiente:

1- El usuario, a través de la página web de su servidor de autenticación y utilizando su navegador web, normalmente ya tiene una página web abierta en dicho servidor, solicita el "Logout", haceindo clic en el botón o link correspondiente.

2- El servidor de autenticación responde al navegador web, si todo es correcto, enviándole una lista de URLs, equivalente a la que devuelve en caso de "logging", pero en este caso las URLs incluyen la orden LOGOUT. Y si el servidor de autenticación tiene registrado que ese usuario ha usado el cargador de credenciales para aplicación, añade a la lista una URL que es enviada al mismo servidor de autenticación solicitando el lanzamiento del cargador de "cookies" de aplicación pero con la acción LOGOUT.

3- Para cada URL el navegador web realiza la consulta y, por tanto, comunica la orden de LOGOUT a cada uno de los PoAs o GPoAs registrados. Y además una de las consultas es para solicitar al servidor de autenticación la ejecución del cargador de "cookies" de aplicación con la acción LOGOUT.

4- El servidor de autenticación devuelve, como respuesta al lanzamiento del "Loader", una respuesta HTTP tipo JNLP, tal y como se muestra en la Fig. 4-15, con la orden de lanzar el Loader para LOGOUT.

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="http://fudaqs2.ciemat.es/PAPI/Loaders/java">
 <information>
      <title>PAPI Token Loader</title>
      <vendor>TJII Acquisition Group (CIEMAT)</vendor>
      <description>Program for loading PAPI tokens</description>
 </information>
 <security>
      <all-permissions/>
 </security>
 <resources>
      <j2se version="1.5+"/>
      <jar href="papiCookieManager.jar" />
 </resources>
 <application-desc main-class="papiUtils.CookieLoader">
                          <argument>LOGOUT</argument>
```

Fig. 4-15 *Mensaje JNLP enviado por el servidor de autenticación en un proceso de Logout para que el navegador web del usuario lance el Loader que gestiona el repositorio de "cookies" de aplicación*.

5- El "Loader" es lanzado en el cliente y elimina las "cookies" correspondientes a la federación, del repositorio de "cookies" de aplicación, y además bloquea dicho repositorio para que no se puedan cargar más "cookies" PAPI hasta una nueva acción de LOGING.

A partir de este punto, cualquier aplicación que haga uso del repositorio de "cookies" de aplicación, no podrá incluir "cookies" PAPI en sus consultas HTTP, ya que éstas han sido borradas del repositorio.

4.7. Integración de servicios EFDA

Uno de los principales objetivos a la hora de implantar una infraestructura de seguridad inter-organización en EFDA era facilitar el acceso seguro a servicios comunes y fomentar el uso de este tipo de entornos colaborativos a partir de las facilidades que da la federación a nivel de identificación de usuarios. En este sentido la

federación ha demostrado servir como punto común de acceso a los diferentes servicios EFDA y ha simplificado la tareas de registro y control de acceso a los diferentes servicios.

4.7.1. Paso de información del usuario a las aplicaciones

Una de las funciones más importantes en un sistema de control de acceso es el paso, a la aplicación o servicio que está siendo protegido, de los datos del usuario que está intentando acceder. Gracias a esta información el servicio o aplicación puede:

- Realizar funciones de registro: El servicio puede comprobar si dicho usuario está ya registrado como usuario del servicio y en caso contrario, bien indicarle a éste la necesidad del registro, o bien iniciar un proceso de registro para dicho usuario. Lo que en este tipo de sistema de control de acceso se suele llamar un proceso de invitación al usuario.
- Realizar controles de acceso adicional.
- Iniciar automáticamente una sesión personalizada para el usuario, por ejemplo, en función de la organización a la que pertenece.
- Generar estadísticas de acceso más fiables y eficaces.

PAPI implementa principalmente un sistema de paso de información del usuario que resulta válido, tanto para recursos locales al servidor Apache de PAPI, como para servicios remotos accedidos a través de la directiva de configuración "Remote_URL" y el módulo de "proxy" inverso de PAPI. El método se basa en la inclusión de una cabecera HTTP propia "X-PAPI-Hcook" en la consulta. En ella se envía el contenido de la credencial PAPI "Hcook" en la que se incluye la información que el punto de acceso dispone del usuario. Como ya se ha explicado previamente, la información que le llega a un punto de acceso no tiene que ser toda la que la infraestructura, y más concretamente el servidor de autenticación o el GPoA, conoce del usuario, ya que PAPI implementa mecanismos de reescritura y anonimización de dicha información. La anonimización empleada permite mecanismos de rastreo de la actividad del usuario, pero dichos mecanismos requieren de la participación de las 3 partes: servidor de autenticación, el GPoA o PoA que realizó la anonimización y el servicio implicado. De esta forma se garantiza por un lado el anonimato del usuario, y por otro lado, la

capacidad de poder trazar las acciones realizadas por un usuario en caso de necesidad. Las directivas PAPI involucradas en estas tareas son:

- *User_Data_Passthrough*: Incluida en la configuración de un punto de acceso permite el paso de información del usuario al recurso a proteger. La información se pasa a través de una cabecera "X-PAPI-Hcook" de tipo HTTP que es incluida dentro de la consulta.
- *GPoA_Hash_User_Data*: Si se asigna a 1, el GPoA realiza una función hash sobre el identificador de usuario, lo que permite anonimizarlo antes de ser pasado al PoA hijo.
- *Hash_User_Data*: Si se asigna a 1, el punto de acceso anonimiza el identificador de usuario mediante una función hash, antes de pasar la información al recurso que está siendo protegido por dicho PoA.
- *GPoA_Rewrite*: Permite definir, en un GPoA, reglas de reescritura mediante expresiones regulares que permiten traducir la información del usuario antes de ser enviada a un PoA hijo.
- *User_Data_Rewrite*: Permite definir, en un PoA, reglas de reescritura mediante expresiones regulares que permiten traducir la información del usuario antes de ser pasada al recurso que está siendo protegido por dicho PoA.

La información del usuario se transmite utilizando el siguiente formato:

<Datos de usuario>::<id de usuario>@<id del servidor de autenticación>%<IP fuente del usuario>%

Los campos incluidos son los siguientes:

- *Datos de usuario*: Cadena de caracteres que conforman una lista formada por elementos del tipo "nombre del campo=valor" separados por ",".

- *Id del usuario*: Identificador del usuario en el servidor de autenticación o en el GPoA (si ha sido anonimizado por este)

- *Id del servidor de autenticación*: para conocer cuál es el servidor de autenticación asignado al usuario.

- *IP fuente del usuario*: La IP fuente desde donde se conecta el usaurio.

4.7.2. Servicio Wiki de EFDA

4.7.2.1. Introducción

El entorno wiki de EFDA se utiliza como repositorio documental. En dicho entorno los usuarios pueden definir áreas y en cada área pueden definir nuevos temas de discusión. Cada tema de discusión tiene asociado una o varias páginas web y cada una de ellas admite opiniones de otros usuarios. Las páginas web del sistema wiki son documentos vivos que pueden ser actualizados y ampliados continuamente, por lo que resulta muy útil para estructurar opiniones y discusiones entorno a documentos estructurados en formato web. Además pueden ser incorporados al sistema documentos externos y contenidos en formato digital, que son enlazados a las páginas web mediante links HTML, lo que lo convierten en un potente y flexible repositorio documental colaborativo. Cara al usuario, el entorno wiki de EFDA incorpora un editor vía web de tipo WYSIWYG (What You See Is What You Get) para el contenido de la página, de tal forma que según lo ve el usuario al editarlo, así se ve luego como página web final, por lo que evita la necesidad de tener que aprender HTML y hojas de estilo para poder producir documentos web con un formato bastante rico. La web resultante es totalmente coherente en estilo y en menús de navegación. Desde el punto de vista de la gestión wiki mantiene un sistema de permisos basado en usuarios y grupos de tal forma que se puede restringir el acceso a diferentes áreas y funcionalidades del entorno wiki.

4.7.2.2. Arquitectura del sistema

Desde un punto de vista técnico el sistema wiki de EFDA se basa en el software de código abierto XWiki [49]. Este sistema está desarrollado en JAVA y tiene como arquitectura base la utilización de Servlets para la gestión de las consultas HTTP. En su distribución XWiki dispone de dos opciones. Una basada en la utilización de como JETTY [50] como motor de servlets, lo que permite la instalación del sistema en modo autocontenido, es decir, sin necesitar ningún componente adicional, y otra basada en la utilización de Tomcat como motor base de servlets, y que por tanto tiene que estar preinstalado.

En el caso del servicio wiki de EFDA, éste se basa en la utilización de Tomcat como motor servlets sobre el que se ejecuta la aplicación "xwiki". Este servicio utiliza

una base de datos mysql como repositorio de almacenamiento de toda la información que maneja: documentos, plantillas, usuarios, grupos, enlaces, etc. A nivel de comunicaciones e interfaces, las conexiones de este servicio wiki se realizan a través de protocolo HTTPS.

Basado en la experiencia con el sistema de participación remota del CIEMAT, que también utiliza Tomcat como soporte para muchas de sus aplicaciones, se decide integrar PAPI utilizando una arquitectura estándar con el servidor Apache como fron-end del sistema. En la Fig. 4-16 se muestra la arquitectura de la solución. En ella se muestra el sistema ya existente en color blanco y los componentes nuevos en color verde. En el diagrama aparece Apache como fron-end de todas las conexiones HTTP (en este caso HTTPS), y como back-end, el servicio ya implantado y basado en el servidor Tomcat con la aplicación "xwiki" como núcleo del sistema.

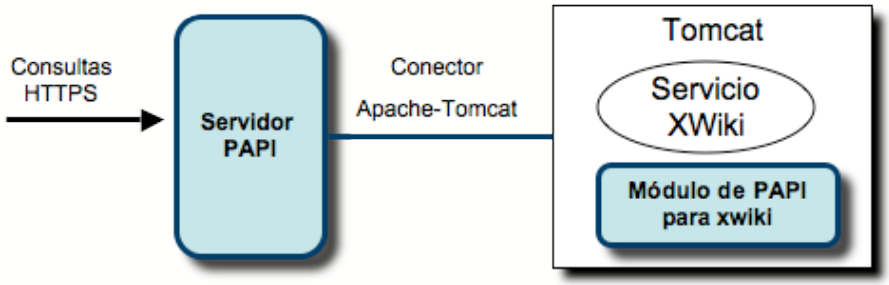

Fig. 4-16 *Arquitectura de la solución en la que se integra el servicio wiki de EFDA en la federación*

En la solución presentada se consigue preservar la arquitectura previa sin introducir ninguna modificación en su funcionamiento ni configuración. Como ya se comentó en el punto referente a la configuración de PAPI a nivel de organización, el sistema instalado en EFDA se basa en un servidor PAPI donde se implementan y configuran todos los puntos de acceso con el fin de tocar lo menos posible los sistemas originales a proteger. La conexión entre el servidor PAPI y la plataforma Tomcat del servicio Wiki se implementa mediante el conector Apache-Tomcat, como módulo "jk" de Apache, y desarrollado dentro del proyecto Jakarta.

4.7.2.3. Autorización basada en PAPI

En referencia a la gestión de usuarios, wiki mantiene una base propia de usuarios registrados que almacena en una tabla de su base de datos MYSQL. Con el objetivo de modificar lo menos posible el código y arquitectura del sistema, se decide realizar la implementación de un módulo plug-in para "xwiki" que sirva como método de autenticación para el servicio wiki de EFDA. Para ello "xwiki" habilita un sistema de módulos basado en implementar la interface "XWikiAuthService". Finalmente, se toma la opción de en vez de realizar una implementación desde cero de la interfaz, se opta por extender la clase "XWikiAuthServiceImpl", que como muestra la Fig. 4-17, incluye una primera implementación de los métodos de la interfaz: "authenticate", "checkAuth", "findUser", "getAuthenticator" y "showLogin". Para el caso de módulo de autorización para PAPI, la nueva clase "PapiAuthImpl" se limita a sobrescribir el método "checkAuth", dejando el resto intactos ya que la idea es que todo el proceso de autenticación sea igual que si el usuario hubiera introducido su login y password, sólo que en este caso el Login va a ser proporcionado por PAPI y el chequeo de la password no se va a realizar, por lo que si un usuario está registrado en "xwiki", tendrá acceso automático siempre que acceda a través de PAPI.

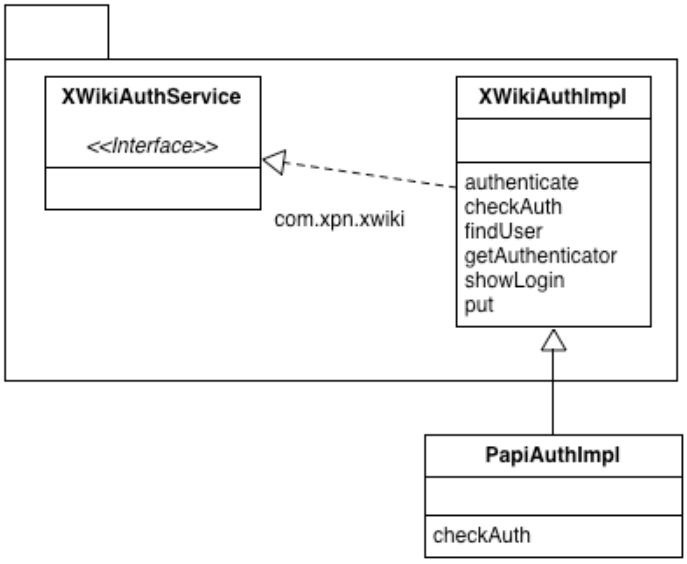

Fig. 4-17 *Diagrama de clases del módulo de autenticación PAPI para "xwiki"*

Las directivas de configuración de los módulos de "xwiki" se asignan en el fichero "xwiki.cfg". Para el nuevo módulo de autenticación de PAPI las directivas a configurar son:

- xwiki.authentication.papi: Si se fija a 1, activa el módulo de autenticación de PAPI para "xwiki".
- xwiki.authentication.papi.IP: Sirve para definir la dirección IP del servidor PAPI que controla el acceso al servicio.
- xwiki.authentication.papi.userRegex: Sirve para definir una expresión regular que permita extraer el identificador de uaurio "xwiki" de la información de usuario que es enviada por PAPI a través de la cabecera "X-PAPI-Hcook".

La Fig. 4-18 muestra el diagrama de actividad del módulo. En primer lugar, chequea si en la configuración está activa la autenticación PAPI, en cuyo caso, se procede a verificar si la IP del servidor PAPI, a través del cual pasa la información el usuario, es correcta. A continuación se extrae el identificador de usuario para el servicio wiki de EFDA, a partir de al información remitida por PAPI, y con dicho identificador, se procede a verificar si el usuario está registrado en el servicio wiki de EFDA. Si está registrado, el módulo da la autenticación por correcta y devuelve el objeto "XWikiUser" correspondiente al sistema.

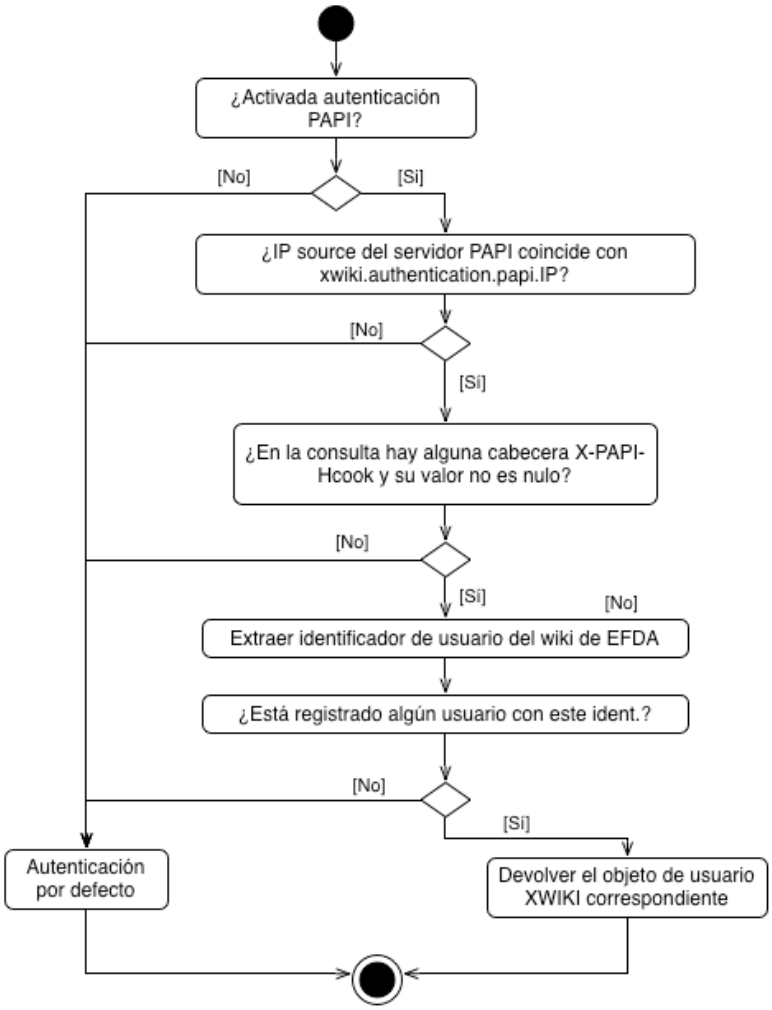

Fig. 4-18 *Diagrama de actividad del módulo de autenticación PAPI para "xwiki"*

4.7.3. Sistema "e-Survey"

4.7.3.1. Introducción

El sistema "e-Survey" es un servicio que permite organizar encuestas y cuestionarios en un sistema web, para que los usuarios puedan completarlo de forma remota utilizando un navegador web como aplicación cliente. El sistema, desde el punto de vista del administrador, provee de herramientas vía web que permiten, de forma sencilla, diseñar e implementar diferentes cuestionarios con una gran variedad de tipos de preguntas. Así mismo, el sistema dispone de un entorno donde usuarios con ciertos

permisos pueden generar informes y estadísticas a partir de los resultados del cuestionario, así como exportar dichos datos en una gran variedad de formatos.

Este entorno está siendo utilizado actualmente en el grupo de participación remota de EFDA para la gestión de un cuestionario llamado "RP Facilities" y que consiste en un conjunto de 126 cuestiones agrupadas en 11 categorías y cuyo objetivo es disponer de información, de diferentes organizaciones de EFDA, relativa a: infraestructura de comunicaciones, sistemas de acceso a datos y a equipos remotos, equipamiento de videoconferencia, y aspectos relativos a la infraestructura de seguridad. Gracias a esta información se dispone de una visión global a nivel de EFDA de las infraestructuras relacionadas con la participación remota en los laboratorios de fusión. Gracias a su implementación en el sistema "e-Survey" los responsables de cada organización pueden mantener la información actualizada y disponible. Además del "RP Facilities" se han implementado otros cuestionarios relativos al uso y la percepción de las tecnologías de participación remota por parte de los usuarios.

4.7.3.2. Arquitectura del sistema

El sistema "e-Survey" se ha implentado mediante la tecnología "Limesurvey" [51]. Esta tecnología se basa en una arquitectura tipo aplicación web, soportada por un servidor Apache que gestiona las consultas HTTP y la ejecución del motor de interpretación de PHP, y una serie de módulos PHP que conforman la aplicación final. El sistema utiliza como repositorio de almacenamiento un sistema de base de datos MySQL con el que se conecta a través de su interfaz de acceso remoto TCP. En dicha base de datos el sistema "e-Survey" almacena, tanto datos relativos al diseño e interfaz de los cuestionarios, como las respuestas a las diferentes cuestiones introducidas por los usuarios. Adicionalmente, a nivel de seguridad, "e-Survey" maneja por un lado, usuarios del sistema con permisos para gestionar y administrar diferentes cuestionarios, y por otro lado, para cada cuestionario que requiere acceso controlado, existe una tabla "tokens", donde se registran los usuarios que tienen acceso a introducir o modificar datos de dicho cuestionario.

La implementación del sistema se basa en módulos codificados en lenguaje PHP que conforman la aplicación final. Además de los módulos que implementan diferentes casos de uso de los usuarios: añadir un cuestionario, editar un cuestionario, añadir

consultas, rellenar un cuestionario, etc., existen otros módulos que son importados por los primeros (generalmente al comienzo de su ejecución) y que realizan funciones básicas y transversales: acceso a parámetros de configuración de la aplicación, funciones de idiomas y de traducción, funciones de acceso a la base de datos, funciones de control de acceso, etc. Este esquema de implementación sin mantener el estado del sistema requiere mecanismos que permitan mantener el estado de ejecución de cada sesión de usuario. En este acaso, el mecanismo que utiliza "e-Survey" se basa en "cookies" de sesión PHP.

4.7.3.3. Módulo de PHP para PAPI

Uno de los resultados de la integración de "e-Survey" en la federación EFDA, es la implementación de un módulo que permitiera a cualquier aplicación PHP incluir los datos provenientes del usuario a través de PAPI. Con el objetivo de ser una solución lo más general posible, el módulo devuelve los datos del usuario obtenidos de PAPI en un vector tipo asociativo (tipo "Hash") en el que se define una serie de entradas comunes a cualquier esquema de atributos del usuario como son:

- "UserId": identificador del usuario asignado por PAPI
- "AS_Server": Identificador del servidor de autenticación donde se ha autenticado el usaurio,
- "IP": IP origen del usuario que originó la consulta.

Y otra serie de entradas que se corresponden con los identificadores de atributos incluida en la aserción PAPI, de tal forma que cada "Atributo=valor" se corresponde con un $PAPI["Atributo"]=valor.

Adicionalmente y cara a hacer el módulo más flexible se incluye un esquema de traducción del nombre de los atributos para que puedan ser acoplados a los valores que maneje la aplicación.

4.7.3.4. Autorización basada en PAPI

"e-Survey" y más concretamente su tecnología base "Limesurvey" no implementa ningún mecanismo que permita la incorporación de nuevos sistemas de autorización sin necesidad de modificar el código original. Derivado de este hecho, se realizó un análisis

del código original y se ha optó por realizar una modificación con el menor impacto, y preservando la mayor cantidad de código original, que fuera posible, de tal forma que resultara sencilla la actualización del sistema a nuevas versiones si así fuera necesario, es decir, priorizando la mantenibilidad del sistema. Existen dos módulos principales "index.php" y "admin.php" que sirven como puntos comunes de acceso a todas las funcionalidades de la aplicación.

El módulo "index.php" se utiliza como punto de acceso a todas las funcionalidades relativas al uso de cuestionarios ya creados tales como: listar los cuestionarios activos con acceso por parte del usuario, rellenar un cuestionario, guardar las modificaciones. o imprimir los datos introducidos. En función de su actividad, se toma la decisión de insertar las funciones relativas a PAPI en el punto donde el sistema detecta que no existe una sesión PHP creada para este usuario e intenta crear una a partir de datos que le son suministrados del usuario. En este caso suministrados a través de PAPI.

El módulo "admin.php" se utiliza como punto de acceso para todas las funcionalidades relativas a la gestión de cuestionarios tales como: crear nuevos cuestionarios, modificar cuestionarios, registrar usuarios y gestionar sus permisos, gestionar las plantillas de presentación de los cuestionarios, etc. Al igual que con el módulo "index.php se insertan las funciones relativas a PAPI en el punto donde el sistema detecta que no existe una sesión PHP creada para este usuario y se comprueba si existe información suministrada a través de PAPI.

4.8. Extensión a nuevos desarrollos y protocolos

Las infraestructuras de autenticación y autorización multi-organización tipo PAPI, Shibboleth o OpenID, están muy orientadas a recursos web y al navegador web como aplicación cliente. A partir de aquí, las diferentes tecnologías se han ido extendiendo, consiguiendo integrar diferentes servicios HTTP, pero utilizando siempre el navegador web como aplicación cliente del usuario. PAPI, como se ha descrito en el punto 4, ha sido satisfactoriamente integrado con otro tipo de aplicaciones cliente, en este caso aplicaciones JAVA, que utilizan HTTP como protocolo base para sus conexiones. Aunque, la solución se ha desarrollado únicamente con aplicaciones desarrolladas en lenguaje JAVA, la tecnología es fácilmente extensible a otros lenguajes.

Aparte de los sistemas basados en HTTP, en la comunidad de fusión existen tecnologías de acceso a datos ampliamente utilizadas y con gran relevancia en el entorno de la participación remota, pero que su conexiones no están basadas en el protocolo HTTP. Estas tecnologías requieren asimismo de una infraestructura de autenticación y autorización distribuida entre organizaciones para facilitar el acceso de los usuarios de forma segura , y sin condicionamientos relativos a la IP origen desde donde se conectan. De ahí que desde la propia comunidad de fusión se demande la integración en la federación EFDA de servicios y tecnologías no basadas en conexiones HTTP. Con este objetivo se ha desarrollado una solución para PAPI, basada en tokens de servicio. Dicha solución, junto con dos casos reales donde se ha aplicado con éxito, son descritos a continuación.

4.8.1. La solución "Service Token" en PAPI

La integración de cualquier tipo de tecnología en la federación EFDA exige el cumplimiento de ciertos requisitos principales:

- Soportar redireccionamiento: Ya que la arquitectura de PAPI en la federación EFDA se basa en el esquema de agrupación de PoAs mediante elementos GPoA, y esta solución requiere la redirección del usuario hacia el GPoA padre hasta conseguir su identificación.
- Conservar el carácter distribuido de la tecnología PAPI: Este requisito garantiza la fácil gestión de la federación, ya que por un lado los usuarios son gestionados en sus organizaciones origen, mientras que los recursos son gestionados por sus organizaciones propietarias. Con ello se consigue que, por un lado, nuevos usuarios o nuevos servidores de autenticación no interfieran en el resto de organizaciones, y por otro lado, que nuevos servicios o recursos tampoco interfieran en el funcionamiento del resto de organizaciones de la federación.
- Integrarse en el sistema Single-Sign-On de la federación EFDA. Con ello se exige, por un lado, que con una sola autenticación el usuario sea capaz de ser identificado por cualquier recurso federado, y por otro lado, que la solución debe ser lo más transparente posible para el usuario.

- Cualquier interacción con el usuario relacionada con la autenticación debe circunscribirse al entorno del servidor de autenticación de su organización. Ya que es el entorno en el que el usuario confía y en el exclusivamente debe confiar.

La idea de utilizar tokens de servicio para la conexión entre un cliente y un servicio no es nueva y es la base tecnológica de sistemas Kerberos (descrito en detalle en el punto 3.2.2). Básicamente, la solución se basa en que si un cliente quiere establecer una conexión con el servidor, debe adjuntar a la conexión un componente adicional (llamado token se servicio). El servidor, gracias a ese token de servicio, puede, bien identificar al cliente de la conexión, bien confirmar que dicho cliente tiene acceso al servicio.

En el caso de la aplicación a PAPI, tal y como se muestra en el diagrama de colaboración de la Fig. 4-19, el proceso es el siguiente:

1. Una aplicación quiere realizar una conexión contra un servidor protegido debe obtener de la infraestructura PAPI (en este caso de la federación) un token de servicio temporal (que expire en poco tiempo y que no pueda ser reutilizado) para poder incluir dicho "token" en la conexión. La aplicación solicita a la infraestructura PAPI, y más concretamente a un PoA configurado como proveedor de tokens temporales asociados al servicio al que la aplicación cliente desea realizar la conexión, un "token" para dicho servicio. La solicitud del "token" a la infraestructura PAPI se realiza a través de una conexión HTTP por lo que, en este punto, la solución utiliza tecnología de autenticación y autorización estándar de PAPI, y por tanto, se pueden utilizar todos los mecanismos que PAPI dispone para definir policitas de acceso.
2. Si la solicitud de "token" de servicio lo requiriera, al ser una consulta estándar HTTP a un recurso protegido y federado, puede que sea redireccionada al GPoA de la organización o incluso el GPoA de la federación, de forma transparente para el usuario. Una vez resueltas todas las redirecciones, el PoA proveedor devolverá una respuesta de error, en caso de que el usuario no disponga de los permisos necesarios, o una respuesta afirmativa, en caso contrario. Si el usuario dispone de los permisos necesarios, el PoA proveedor devuelve a la aplicación cliente un "token" temporal para el servicio solicitado. Dicho "token" contiene información del usuario, suministrada por PAPI, y que puede ser utilizada por el servicio destinatario. Además, el "token" está cifrado con clave simétrica, que el

PoA proveedor comparte con el servicio que autoriza. Esto se hace para que exista una relación unívoca entre cada PoA proveedor y el servicio al que da acceso.

3. La aplicación obtiene el "token" solicitado y puede utilizarlo, antes de que expire, para establecer la conexión con el servicio. El esquema de intercambio del "token" en la conexión no está definido, y por tanto se adapta a la solución aplicada. En los casos descritos en los puntos siguientes, "MDSplus" aprovecha un campo usuario que es enviado en el protocolo de conexión, mientras que en el caso del sistema de monitorización en remoto, el "token" se envía como primeros datos de la conexión.

4. El servicio al recibir la petición de conexión y obtener el "token" temporal de servicio, tiene dos opciones, contempladas dentro del esquema desarrollado. La primera es descifrar y verificar la validez del "token" temporal. Y la segunda consiste en realizar una consulta de verificación a la infraestructura PAPI, en concreto al PoA proveedor del "token".

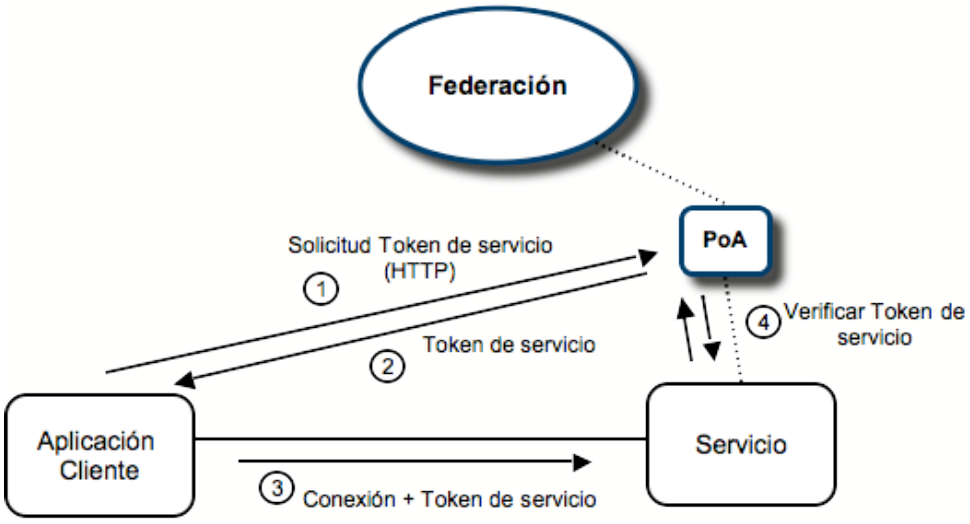

Fig. 4-19 *Diagrama de colaboración del esquema de autorización basado en "token" temporal de servicio aplicado a PAPI*

4.8.1.1. PoA proveedor de tokens temporales de servicio

El proveedor de tokens temporales de servicio que implementa PAPI se basa en un componente PoA estándar al que se le incorpora un servicio de tokens temporales, y de hecho (como se muestra más adelante) en la configuración del PoA esta capacidad no supone más que una etiqueta, compatible con el resto de directivas posibles en un PoA. Esto lo que supone a nivel funcional, es que el esquema de autenticación y autorización para tokens de servicio es completamente compatible con el que se utiliza para cualquier otro servicio web a controlar, y por tanto, en un PoA proveedor de tokens de servicio se pueden definir GPoAs padres, se pueden definir reglas de acceso, etc.

Dentro de lo que es el diagrama de actividad de un PoA, la inclusión de esta funcionalidad tiene dos bloques: uno destinado a chequear la validez de un token enviado por un servicio para su comprobación, y otro, destinado a generar un nuevo token de servicio, si el PoA incluye en su configuración la directiva "Service_Token".

Función de comprobación de tokens de servicio

La función de comprobación de tokens de servicio se resuelve antes de realizar ningún tipo de control de acceso. La consulta posee el formato:

- STOKEN: Donde se incluye el token de servicio, en formato texto y codificado como "Base64", tal cual le fue entregado por la aplicación cliente.
- SNAME: Identificador del servicio que realiza la petición de comprobación.

A continuación se descifra el contenido del campo STOKEN y se comprueba:

- Si el token va destinado al servicio que realiza la consulta.
- Si el token no ha expirado.

Si existe algún tipo de error, se responde con: "PAPIERROR, Error on test". En caso contrario devuelve: "PAPIOK, Service Token OK"

Función de generación de tokens de servicio

Esta función se realiza una vez el PoA ha pasado por todas las fases relacionadas con la identificación del usuario y el filtrado de las reglas de control de acceso

configuradas. En este caso, el PoA, en vez de gestionar la consulta dejándola continuar, que es la acción por defecto, genera y envía un token de servicio a la aplicación cliente:

Crea un campo en formato texto con los parámetros:

- Hora actual.

- Identifiaodor del PoA generador del token de servicio.

- Identificador del servicio para el que va destinado

- Y una aserción en formato texto con información relativa al usuario y cuyo formato puede ser configurado mediante la propiedad "Format" de la etiqueta "Service_Token" en el fichero de configuración de PAPI.

El campo es cifrado utilizando el cifrado simétrico AES de 192 bits, con la clave definida como la propiedad "key" de la etiqueta "Service_Token" en el fichero de configuración de PAPI. Y el resultado es incluido como el valor del atributo "Stoken" en un mensaje HTTP de respuesta a la aplicación cliente que solicita el token de servicio. El formato de dicho mensaje es: "PAPIOK,Stoken=<campo cifrado>".

4.8.1.2. El agente de token de servicio de PAPI

La aplicación cliente que utiliza el esquema de autorización asado en token de servicio, antes de realizar una conexión "TCP" o "UDP" tendrá que obtener el token apropiado al servicio al que se quiere conectar. Para ello, y según el esquema planteado, tendrá que realizar una petición HTTP solicitando el token de servicio a un PoA concreto. El objetivo es que esta petición HTTP tiene que ser compatible con el esquema de autenticación y autorización de PAPI. Para ello debe ser capaz de usar el repositorio común de "cookies" PAPI para aplicaciones (descrito en el punto 3.7.1) que es utilizado por las aplicaciones JAVA y esta integrado en el esquema Single_Sign_on de PAPI a través del cookieLoader.

Para conseguir el objetivo antes planteado existen dos vías. La primera consiste en dejar a la aplicación, y a su desarrollador, la responsabilidad de utilizar librerías HTTP compatibles con PAPI y capaces de acceder al repositorio de "cookies" comunes. Eso obligaría a desarrollar nuevas APIs de acceso para otros lenguajes de programación aparte de JAVA. Y estar continuamente actualizando y creando diferentes APIs por cada nuevo lenguaje de programación en el que se desarrolle una aplicación cliente. La

segunda vía consiste en desarrollar un elemento intermediario capaz de atender las consultas de la aplicación que lo requiera de una forma sencilla, y que por otro lado, sea capaz de pasar dichas consultas al PoA correspondiente, utilizando librerías HTTP con acceso al repositorio único de "cookies" y compatibles con PAPI.

Finalmente, por mantenibilidad y facilidad de integración, además de otras ventajas derivadas del uso de un componente configurable en el entorno PAPI, se optó por el desarrollo de un agente que se mantiene activo en el equipo cliente y gestiona todas las peticiones de token de servicio, pasándoselas a los PoAs correspondientes y utilizando para ello librerías HTTP compatibles con PAPI.

El agente de tokens de servicio, al realizar una función de "proxy" de peticiones locales, se ha implementado siguiendo un diseño estándar de servicio "TCP" "multi-thread". Dicho diseño se basa en un bloque principal cuya función es esperar por nuevas conexiones "TCP", y cada vez que se establece una nueva, se comprueba que provenga de un proceso local al equipo (es un servicio local al sistema), y si todo es correcto, se inicia un nuevo "thread" al que se le asigna la gestión de dicha consulta. Este modelo "multi-thread" permite paralelizar las consultas recibidas concurrentemente dotando a la solución de cierto grado de paralelismo a nivel de ejecución. La solicitud de "token" de servicio enviada por la aplicación cliente es transferida al PoA correspondiente a través de una consulta HTTP compatible con PAPI, y la respuesta del PoA es comunicada a la aplicación cliente, dándose por concluida la gestión de la consulta y finaliza la ejecución del thread correspondiente, con la consiguiente liberación de recursos del sistema para posteriores solicitudes.

Desde el punto de vista de la implementación el sistema lo forman dos clases principales. La clase "STAgent" implementa el bloque principal del servicio, encargándose de esperar por nuevas solicitudes a través de conexiones al "socket" tipo servidor. Así mismo, la clase proporciona el método "SendMessage" para generar consultas para el agente con el formato adecuado. Una vez recibida una nueva consulta, utiliza la clase "STRequest" (que implementa la interfaz "Runnable" para la implemntación de "threads" en JAVA) para gestionar dicha consulta desde ese punto de la ejecución hasta que se da por resuelta. "STRequest" utiliza la clase "ServiceToken", y más concretamente su método "requestTokenToServer" para resolver la consulta.

La clase "ServiceToken" es sobre la que recae todo el peso de la gestión de "tokens" de servicio. En ella se encapsulan todos los métodos que una aplicación puede necesitar para utilizar dichos tokens:

- requestTokenToServer: Método que implementa la solicitud de un token de servicio a un PoA.
- validateWithKey: Método que implementa la validación de un token de servicio de forma local, es decir, sin recurrir a consultas al PoA que lo generó. Para poder ser utilizado, se debe poseer la clave con la que el token de servicio fue cifrado por el PoA que lo generó.
- validateWithServer: En este caso, y a diferencia de la validación en local, la validación se realiza mediante una consulta al PoA que generó el token. Es un método muy cómodo ya que evita el tener que compartir claves.

Aprovechando el esquema de carga de credenciales para aplicaciones JAVA a través del cookieLoader y la tecnología JWS, se incorpora a dicho "Loader" una funcionalidad que, además de precargar las "cookies" PAPI en el repositorio común de "cookies" para aplicaciones, permite iniciar el agente de tokens de servicio en el equipo cliente. Este servicio permanecerá activo hasta que una fase de "logout" se apaga mediante el envío de la orden correspondiente por parte del "Loader". Todo el código referente al agente de "tokens" de servicio se encuentra incluido en la librería "papiCookieManager" y por tanto el código está convenientemente firmado para ser utilizado con todas las garantías a nivel de seguridad del equipo cliente.

4.8.2. MDSPlus

El acceso a datos es uno de los aspectos más importantes en el área de la participación remota en fusión, y "MDSplus" [52] se ha convertido en una de las tecnologías más importante en dicho área. "MDSplus" se ha convertido en un estándar de facto para la obtención de datos generados en experimentos de fusión, permitiendo un acceso homogéneo a los mismos independientemente del experimento en el que fueron generados. Gracias a "MDSplus" los investigadores disponen de un conjunto amplio de métodos de acceso a estos datos, que pueden ser utilizados desde una gran variedad de lenguajes de programación (C, PHP, JAVA, FORTRAN), y desde diversos entornos como "MatLab" e "IDL". Otra importante característica de "MDSplus" es que

es posible el acceso a los datos en remoto a través de Internet, permitiendo a los investigadores acceder desde su organización a datos generados en experimentos de otros laboratorios.

Desde el punto de vista de la seguridad en el acceso a los datos, los servidores "MDSplus" instalados, utilizan generalmente el filtrado de conexiones por IP origen. Este mecanismo de control de acceso incluye los problemas típicos: impidiendo de acceso desde fuera del lugar habitual de trabajo e impidiendo la movilidad de los usuarios, no permitiendo la definición de reglas de acceso en base al usuario que accede, estableciendo un nivel de seguridad bajo y garantizando únicamente el origen del acceso, dificultad de gestión en las reglas de filtrado (pudiendo llegar a ser muchas y repetidas). Otro de los mecanismos que implementa "MDSplus" está basado en el esquema de autenticación y autorización desarrollado para el sistema "Globus", que se basa en el empleo de certificados.

La integración de "MDSplus" en la federación EFDA [53] permite la integración del acceso a "MDSplus" en su esquema de autenticación y autorización, lo que evitar los problemas antes comentados y ofrece al usuario una sensación de recurso integrado en su entorno de trabajo, ya que una vez autenticado, éste tiene acceso a todos los recursos "MDSplus" de la federación a los que está autorizado de una forma transparente. Como consecuencia de dicha integración se facilita y potencia el trabajo colaborativo entre los laboratorios federados de fusión.

4.8.2.1. Arquitectura de la solución

A nivel de acceso remoto a datos, "MDSplus" utiliza un esquema cliente-servidor a través de conexiones TCP y basado en un protocolo completamente propietario de "MDSplus", lo que hace imprescindible la utilización de una solución basada en tokens temporales de servicio descrita anteriormente.

El sistema de integración de "MDSplus" en los diferentes lenguajes de programación y entorno de cálculo se realiza a través de la implementación de adaptadores capaces de proveer a un entorno o lenguaje de programación de funcionalidades implementadas en una librería ya compilada. En este caso "MDSplus" implementa en una librería base, desarrollada en "C", toda un conjunto de funciones para el acceso a datos en un servidor remoto. Y para cada uno de lo lenguajes de

programación y entornos matemáticos, tiene implementados wrappers que actúan como interfaces de acceso a las funcionalidades de la librería. Una ventaja muy evidente de este sistema es que si se realizan modificaciones en la librería, éstas repercuten automáticamente en todos los entornos antes descritos, por lo que realizando las modificaciones adecuadas a nivel de la librería de "MDSplus", éstas son automáticamente integradas en cada uno de los lenguajes de programación y entornos de trabajo.

En la Fig. 4-20 muestra un diagrama que representa la arquitectura de la solución. En dicho diagrama se muestran las siguientes secuencias:

1. El usuario se autentica en el servidor de autenticación de su organización. Para ello, envía las credenciales necesarias en función del método de autenticación utilizado en servidor.
2. Si todo es correcto el servidor de autenticación envía al usuario URLs firmadas que contienen aserciones de usuario para cada PoA o GPoA registrado en el servidor de autentciación. Al tratarse un usuario integrado en la federación EFDA, una de las URLs firmadas corresponde al GPoA de la federación EFDA.
3. El usuario solicita el uso de aplicaciones (diferentes al navegador web) y mediante la tecnología JWS se lanza el "Loader" de PAPI que permite obtener y precargar la cookie PAPI del GPoA de la federación. La precarga la realiza en el repositorio de "cookies" PAPI para aplicaciones. Además, el "Loader" deja activo el agente de tokens de sesión "STAgent" a la espera de peticiones por parte de aplicaciones locales.
4. La aplicación utiliza la librería de "MDSplus" que tiene integrado el proceso para solicitar un token de servicio temporal al PoA correspondiente al servidor "MDSplus" a conectar. Para ello realiza una consulta al agente local de tokens de servicio, que se encarga de redirigir la consulta a través de una consulta HTTP y utilizando una librería compatible con el repositorio único de "cookies" PAPI para aplicaciones. Con ello se garantiza que la petición del "token" de servicio seguirá todo el esquema de autorización PAPI utilizado en la federación EFDA, antes de poder obtener el "token" solicitado.

5. Si todo es correcto, el "token" de servicio es devuelto como respuesta del PoA consultado. La respuesta es canalizada a través del "STAgent" y se devuelve, en este caso, a la librería "MDSplus" que solicitó el "token" de servicio.
6. La librería "MDSplus" inicia la conexión "TCP" con el servidor remoto, pasándole como información de usuario el "token" de servicio ontenido previamente.
7. El servidor "MDSplus" realiza la comprobación del "token" de servicio gracias a la clave compartida con su PoA, que es el mismo que originó el "token" de servicio previamente.
8. Si todo es correcto, se completa la conexión.

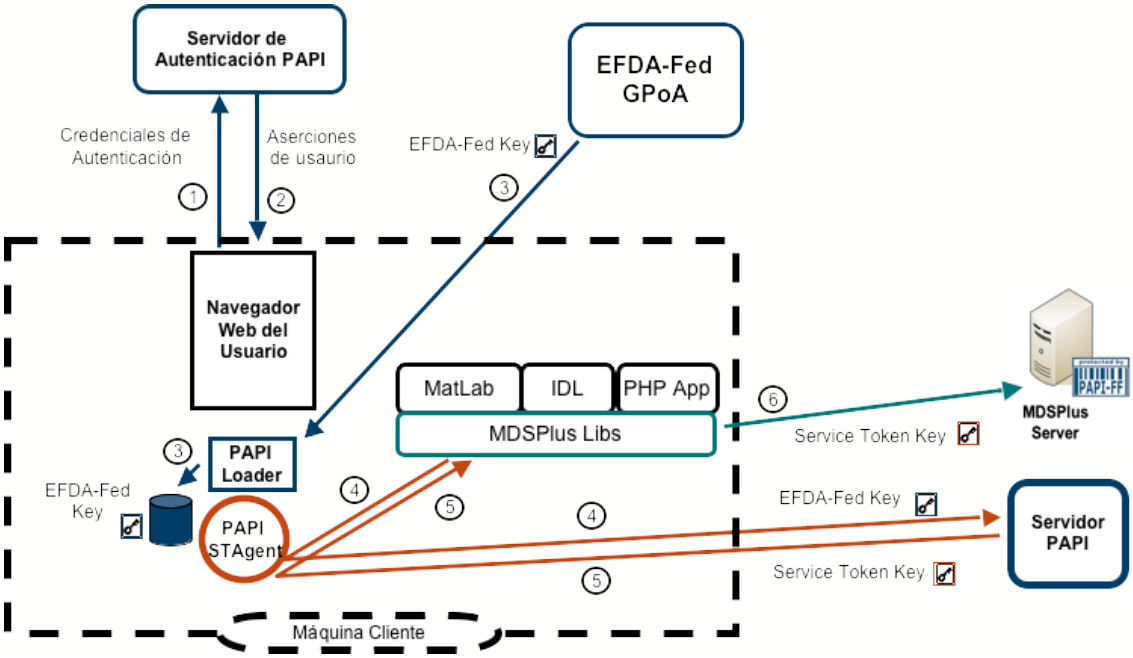

Fig. 4-20 *Arquitectura de la integración de "MDSplus" en la federación EFDA*

4.8.3. Sistema de monitorización en directo

Este proyecto fue desarrollado con el objetivo de crear un sistema de prueba que permitiera la monitorización en remoto de un diagnóstico que se encuentra instalado en JET [54] . La idea era que este sistema sirviera de prototipo para, en el futuro,

desarrollar una infraestructura que permita la instalación de equipos de diagnóstico, y que estos puedan ser monitorizados en remoto durante el desarrollo de experimentos en JET. Este tipo de soluciones van orientadas a usuarios que se encuentran en otras organizaciones, normalmente del entorno EFDA. Un aspecto que resulta clave en este tipo de sistemas es la seguridad, y en ese sentido, se decidió aprovechar la infraestructura que ofrece la federación EFDA para dotar al sistema de un esquema de autorización que englobara, tanto a las organizaciones de los potenciales usuarios del sistema, como a JET como propietaria del servicio. Para realizar la integración del sistema de monitorización online en la federación EFDA, la solución a desarrollar debía de ser compatible con la tecnología PAPI.

4.8.3.1. Arquitectura general de la solución

En la Fig. 4-21 se muestra la arquitectura de la solución, sin tener en cuenta la parte de seguridad. Los componentes de la solución son:

- **Módulo de datos del diagnóstico en JET**: Es un módulo software que se integra en el diagnóstico a monitorizar y es el encargado de dar formato, empaquetar y enviar, los datos que va obteniendo, al servidor de distribución de datos. En el caso concreto del proyecto, el diagnóstico a monitorizar es un sistema de reflectometría de correlación. con frecuencia de muestreo variable en función del ancho de banda de las señales de entrada. El módulo de datos del diagnóstico fue implementado sobre el sistema "LabView" encargado de controlar el diagnóstico.

- **La red local de JET**: esta red se divide en diferentes subredes, entre as cuales están "JETNet" y "JAC". Esto permite el aislamiento a nivel de tráfico y a nivel de seguridad de los diferentes elementos conectados. En el caso concreto del proyecto en cuestión, el equipo diagnóstico y el servidor de distribución de datos se encuentran en subredes diferentes.

- **El sistema de distribución de datos**: Este sistema está encargado de obtener los datos de los diagnósticos y ser capaz de distribuirlos de forma eficiente a cada uno de los clientes que los están monitorizando. Para su implementación, en el proyecto se utilizó una tecnología que se denomina RBNB (Ring Buffering Network Bus) [55] que está siendo utilizada como servidor de distribución de datos del proyecto NEES (Network for Earthquake Engineering Simulation) y en otros proyectos de la NASA.

(National Aeronautics and Space Administration). Como características principales destaca que: permite mecanismos de subscripción tanto a productores como a consumidores de datos, es compatible con prácticamente cualquier tipo de datos, implementa diversos protocolos de conexión, es una solución escalable que permite replica en remoto de ciertos canales de datos entre servidores, y es multiplataforma (está implementada en JAVA).

- **La aplicación cliente de monitorización**: Esta aplicación visualiza todos los aspectos a monitorizar. La zona de visualización está dividida en dos. La primera zona se destina a visualizar el parámetros relacionados con la infraestructura de monitorización tales como: resolución de los datos, ratio de envío de los datos, retardo de la monitorización respecto a la situación del experimento, o información específica del experimento (número, fecha, etc.). La segunda zona muestra datos y parámetros del diagnóstico a monitorizar.

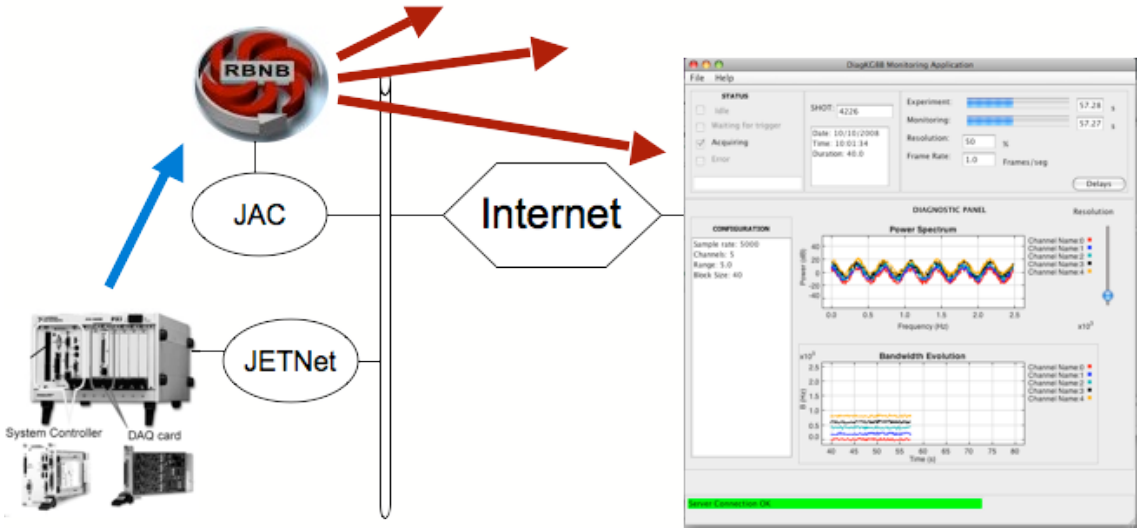

Fig. 4-21 *Arquitectura del sistema de monitorización en remoto y en tiempo real de sistemas de diagnóstico de JET.*

Como conclusiones de la solución cabe destacar su estructura modular que permite sustituir ciertos elementos sin que el resto se vean alterados, y su flexibilidad, que permite la integración de nuevos diagnósticos de una forma sencilla.

4.8.3.2. Integración con PAPI

La seguridad del sistema se va a centrar en la conexión entre la aplicación cliente de monitorización y el sistema de distribución de datos, que es la auténtica interfaz del servicio en el lado de la organización propietaria del servicio. El hecho de que exista una infraestructura de seguridad implantada que cubre a las organizaciones implicadas y que gestiona todos los aspectos referentes a autenticación y autorización, permite focalizar los esfuerzos de desarrollo en otros aspectos del proyecto sin que por ello se vea mermado el nivel de seguridad final del sistema.

La conexión entre la aplicación cliente y el servicio de distribución de datos no es HTTP por lo que hay que optar por el mecanismo de tokens de servicio de PAPI. Este esquema de autorización requiere que la aplicación solicite un token de servicio justo antes de establecer la conexión con el servicio. Desde el punto de la conexión, es necesario comunicar, en algún momento de la misma, el valor del token de servicio desde el cliente al servidor. Por último el servidor debe chequear la validez del token de servicio recibido antes de confirmar la conexión.

RBNB incorpora en su implementación un mecanismo, basado en clases tipo "Interface", que permite incorporar mecanismos de seguridad en las conexiones entre los clientes y el servidor RBNB sin necesidad de modificar el código de la distribución original. Las clases tipo "Interface" que suministra son: "SecurityProviderFactory" y "SecurityProvider". Estas interfaces son implementadas por las clases: "SecurityProviderFactoryPAPIImpl" y "SecurityProviderPAPIImpl" respectivamente. La clase "SecurityProviderFactoryPAPIImpl" implementa la clase "create", requerida por su clase "Interface", en la que únicamente instancia un objeto de la clase "SecurityProviderPAPIImpl", que es el que realmente desarrolla los métodos "clientSideAuthenticate" y "serverSideAuthenticate" que se encargan de implementar todo el mecanismo de autorización entre en la parte cliente y servidor, respectivamente, en la fase de la conexión del cliente al servidor.

En la Fig. 4-22 se muestra la arquitectura general de la integración de PAPI en la conexión entre la aplicación cliente y el servidor de distribución de datos RBNB. Los pasos típicos que se dan en un intento de conexión son:

1. El método clientSideAuthenticate solicita un token de servicio. Para ello utiliza el método "requestTokenToServer" (implementado por la clase ServiceToken y descrito en detalle en el punto 4.8.1).
2. El "STAgent" del equipo local atiende la petición y solicita al PoA asociado al servidor RBNB de JET un "token" de servicio.
3. Si todo es correcto y el usuario está autorizado, el PoA envía un "token" de servicio temporal al "STAgent" que lo solicitó.
4. El cliente recibe el "token" de servicio
5. El método "clientSideAuthenticate" incluye el "token" en el intento de conexión al servidor.

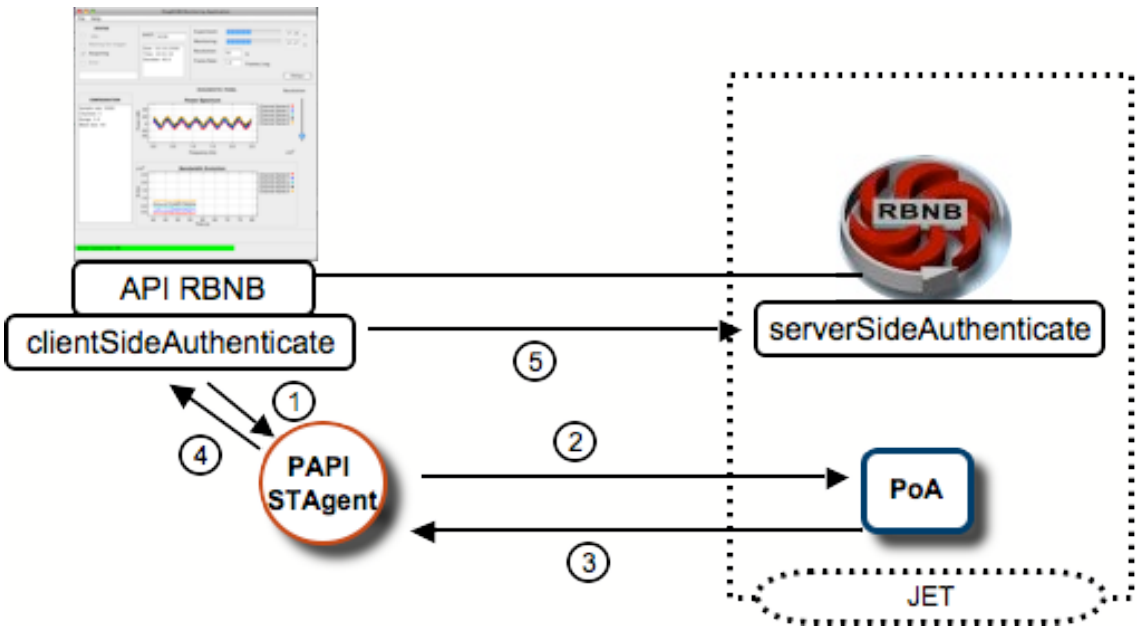

Fig. 4-22 *Arquitectura general de la integración del sistema de PAPI en el sistema de monitorización.*

En cuanto al método "serverSideAuthenticate", también implementado por la clase "SecurityProviderFactoryPAPIImpl", tal como ocurre con la parte cliente, la integración del lado servidor de la conexión utilizando el esquema de token de servicio de PAPI es realmente sencilla, y se resume en los siguientes pasos:

1. Recibir el mensaje desde el cliente y extraer la parte correspondiente al valor del token de servicio.

2. Comprobar el valor de la propiedad de ejecución "papi.ServiceToken.URL" que se corresponde con la URL del PoA asociado al servicio.
3. Si tiene un valor no nulo, se realiza el chequeo del token de servicio a través del PoA asociado, utilizando el método "validateWithServer" que implementa la clase "ServiceToken", ya descrita anteriormente.
4. Si por el contrario, tiene valor nulo, se comprueba el token utilizando una clave, compartida con el PoA asociado, y que se obtiene de la propiedad de ejecución "papi.ServiceToken.key". Para ello se usa el método "validateWithKey" que implementa la clase "ServiceToken".
5. Si la validación no es correcta se lanza una excepción que invalida el intento de conexión.

Además de la implementación de las interfaces, la aplicación de un método de autorización en RBNB requiere que tanto el servidor como la aplicación cliente incluyan dos propiedades en sus entornos de ejecución. La primera es "com.rbnb.authenticate" que debe asignarse a "true", y la propiedad "com.rbnb.securityProviderFactory" que debe asignarse al nombre de la clase que implementa el SecurityProviderFactory, en este caso, "SecurityProviderFactoryPAPIImpl".

Capítulo 5

Conclusiones

Tras una evaluación de posibles alternativas, en el Laboratorio de Fusión por Confinamiento Magnético del CIEMAT, se decidió implantar PAPI como sistema de control de acceso al entorno de participación remota de su máquina ESTELLARATOR TJ-II. Aunque desde un principio el esquema distribuido de funcionamiento y gestión de PAPI se ajustaba a los requisitos del problema, para poder satisfacer todos los requerimientos funcionales y aumentar la eficacia del sistema, se han ido realizado importantes mejoras en PAPI, entre las que destacan:

- La integración de aplicaciones JAVA ya compiladas
- La integración de entornos de control de versión y ejecución como Java Web Start
- El desarrollo de un elemento (GPoA) con capacidad para estructurar los recursos en grupos y para delegar funciones de identificación.
- El desarrollo de un gestor de acceso a recursos con capacidad de definir complejas políticas de control en base a propiedades del usuario y de la consulta.

Todas estas mejoras se han realizando preservando la propiedad de autenticación única (Single-Sign-On) de PAPI, y en la mayoría de los casos, han supuesto innovaciones tecnológicas, no disponibles en sistemas similares como Shibboleth, pero que a su vez podrían ser perfectamente incorporadas a éstos últimos.

El sistema PAPI está funcionando desde el año 2004 como infraestructura de control de acceso a los sistemas del entorno de participación remota del TJ-II, sin que hasta la fecha se hayan detectado fallos en su seguridad.

Como futuras líneas de investigación en este ámbito destacan:

- La integración del esquema PAPI y el esquema de dominio de Windows, de tal forma que un inicio de sesión en uno de ellos sea reconocible y válido para el otro. Esto supondría que si un usuario inicia una sesión Windows en un puesto del CIEMAT estaría automáticamente autenticado en el entorno del TJ-II.
- La integración del nuevo DNIe como sistema de autenticación válido en el TJ-II. Esto supondría una importante mejora en el nivel de seguridad del esquema de autenticación.

En el entorno de fusión, y más concretamente entre las organizaciones de EFDA, existe una creciente necesidad de crear infraestructuras que promuevan y faciliten el trabajo colaborativo y la participación remota entre laboratorios. Como resultado, se ha creado una federación EFDA como estructura multi-organizativa, en la que los usuarios disponen de un entorno seguro con acceso a servicios y recursos, que de forma transparente para ellos, se encuentran distribuidos entre diferentes organizaciones. Como infraestructura de autenticación y autorización soporte a la federación se ha utilizado PAPI, y a nivel de resultados, la nueva federación, ha sido capaz de integrar de forma eficiente servicios HTTP como:

- El sistema de encuesta electrónica vía web del grupo de participación remota de EFDA.
- El servicio Wiki de EFDA.
- El servicio de sistema de ficheros vía web.

Asimismo, y a diferencia de otras, la federación EFDA:

- Dispone de un sistema real de "logout".
- Permite integrar aplicaciones y entornos de ejecución como el JWS.
- Permite centralizar las políticas de acceso en un elemento (servidor PAPI) que actúa como interfaz de entrada a servicios, que son protegidos sin requerir cambios en la instalación o configuración de los mismos.

Además de estas capacidades, conviene remarcar, como innovación la integración de aplicaciones y servicios no HTTP, como es el caso del protocolo "MDSplus" o el sistema de monitorización de diagnósticos en remoto. Proyecto, en el que el desarrollo

pudo centrarse en los requerimientos funcionales, delegando en PAPI todo la complejidad relativa a la seguridad.

Como futuras líneas de investigación, en el ámbito de la federación EFDA, se incluyen:

- Hacer PAPI compatible en un entorno multi-federado que permita a un recurso o a un usuario pertenecer a varias federaciones.
- Integrar un sistema de autenticación delegado, que permita incorporar el esquema de autenticación SecureID de JET como sistema alternativo de autenticación, además del de la propia organización del usuario. Esto permitiría a los usuarios optar a métodos de autenticación de mayor nivel, y así poder acceder a servicios y recursos con mayor niveles de seguridad en el acceso.
- El desarrollo de tecnología que integre lo mejor de cada solución. Lo que permitiría, por un lado, utilizar las novedades incluidas en PAPI, y por otro lado, ser compatible con otros sistemas de seguridad.

Apéndice A

Publicaciones y Congresos

PUBLICACIONES

Autores: R. Castro, D. López
Título: The PAPI system: point of access to providers of information
Publicación: Computer Networks, Volume 37, Issue 6, December 2001, Pages 703-710, ISSN: 1389-1286

Autores: R. Castro, D. López
Título: PAPI, una propuesta de RedIRIS para el acceso ubicuo a recursos de información
Publicación: El Profesional de la Información, Vol 10, Noviembre 2001, Pages 11-14, ISSN: 1386-6710

Autores: R. Castro, D. López
Título: El sistema de control de acceso PAPI
Publicación: Boletín de la red nacional I+D, RedIRIS, Vol 60, Abril 2002, Pages 22-32, ISSN: 1139-207X

Autores: R. Castro
Título: "Avanzando en la seguridad de las redes WIFI"
Publicación: Boletín de la red nacional I+D, RedIRIS, Vol 73, Septiembre 2005, Pages 23-32, ISSN: 1139-207X

Autores: R. Castro, D. R. López and J. Vega
Título: "An authentication and authorization infrastructure: the PAPI system"
Publicación: Fusion Engineering and Design, Volume 81, July 2006, Pages 2057-2061, ISSN 0920-3796

Autores: R. Castro, J. Vega, A. Portas, D. R. López, S. Balme, J.M. Theis, P. Lebourg, H. Fernandes, A. Neto, A. Duarte, F. Oliveira, F. Reis, K. Purahoo, K Thomsen, W. Schiller and J. Kadlecsik.
Título: “PAPI based federation as a test-bed for a common security infrastructure in EFDA sites”.
Publicación: Fusion Engineering and Design. Volume 83, April 2008, Pages 486-490, ISSN: 0920-3796

Autores: R. Castro, J. Vega, A. Portas, A. Pereira, S. Balme, A. Duarte, H. Fernandes, J. Kadlecsik, P. Lebourg, A. Neto, F. Oliveira, K. Purahoo, F. Reis, C. Rodriguez, J. Signoret, J. M. Theis, K Thomsen
Título: "EFDA-Fed: European federation among fusion energy research laboratories"
Publicación: Campus-Wide Information Systems, Volume 25, 2008, Pages: 359 - 373, ISSN: 1065-0741

Autores: R. Castro, J. Vega, A. Portas, A. Pereira, D López
Título: "EFDA-Fed: Una federación internacional para investigación en fusión basada en PAPI"
Publicación: Boletín de la red nacional I+D, RedIRIS, Vol 82-83, Abril 2008, Pages 17-23, ISSN: 1139-207X

Autores: R. Castro, J. Vega, A. Pereira, A. Portas.
Título: "Data distribution architecture based on standard real time protocol".
Publicación: Fusion Engineering and Design, Volume 84, Issues 2-6, Proceeding of the 25th Symposium on Fusion Technology - (SOFT-25), June 2009, Pages 565-568, ISSN 0920-3796

Autores: R. Castro, K. Kneupner, J. Vega, G. De Arcas, J.M. López, K. Purahoo, A. Murari, A. Fonseca, A. Pereira, A. Portas and JET-EFDA Contributors
Título: "Real-Time Remote Diagnostic Monitoring test-bed in JET"
Evento: 7º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 15 - 19 June 2009 , Aix-en-Provence , France
Publicación: Approved for publishing in "Fusion Engineering and Design".

Autores: R. Castro, J. Vega, T. Fredian, K. Purahoo, A. Pereira, A. Portas
Título: Poster, "Securing MDSplus in a multi-organization environment"
Evento: 7º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 15 - 19 June 2009 , Aix-en-Provence , France
Publicación: Approved for publishing in "Fusion Engineering and Design".

CONFERENCIAS (como presentador)

Título: "The PAPI system: point of access to providers of information"
Evento: International TERENA Networking Conference (Networking Middleware), Antalya,
Turkey from 14-17 May 2001

Título: "Ubiquous Internet Access Control: The PAPI System"
Evento: 13th International Workshop on Database and Expert Systems Applications, September 2-6, 2002, Aix en Provence, France

Título: “PAPI Versión 1.5: Sistema de gestión remota e integración con aplicaciones Java"
Evento: Spanish Research Network Technical Meetings 2004, Toledo, Spain, 27-29 Oct 2004

Título: “An authentication and authorization infrastructure: The PAPI system"
Evento: 5th IAEA TM on Control, Data Acquisition, and Remote Participation for Fusion Research, 12-15 July 2005, Budapest

Título: “PAPI based federation as a test-bed for a common security infrastructure in EFDA sites"
Evento: 6º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research 4 - 8 June 2007 , Inuyama, Japan

Título: “EFDA-Fed: Una federación internacional para investigación en fusión basada en PAPI "
Evento: Jorndas Técnicas de RedIRIS 2007, 19-23 Noviembre 2007, Mieres, Spain

Título: “EFDA-Fed: European federation among fusion energy research laboratories "
Evento: International TERENA Networking Conference, 19-22 May 2008, Bruges, Belgum

Título: “Real-Time Remote Diagnostic Monitoring Test-bed in JET"
Evento: "Seventh IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research", 15 - 19 June 2009 , Aix-en-Provence , France

CONFERENCIAS (Contribuciones)

Autores: D. López, R. Castro
Título: Proceeding, “Ubiquous Internet Access Control: The PAPI System"
Publication: "Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on", September 2002, Pages 441-445, ISSN: 1529-4188, ISBN: 0-7695-1668-8
Evento: 13th International Workshop on Database and Expert Systems Applications. DEXA 2002, 2-6 Sept, Aix en provence, France.

Autores: D. López, R. Castro
Título: Proceeding, “Attribute-Based Interactions in a Distributed Authenticatiuon and Authorization Infrastructure"
Publication: "Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on", September 2003, Pages 438-442, ISSN: 1529-4188, ISBN: 0-7695-1993-8
Evento: 14th International Workshop on Database and Expert Systems Applications. DEXA 2003, 1-5 Sept. Prague, Czech Republic.

Autores: R.Castro, D. Lopez, J. Vega
Título: Poster, “An authentication and authorization infrastructure: The PAPI system "
Evento: 5º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 12-15 Jul 2005, Budapest

Autores: R. Castro, J. Vega, A. Portas, D. R. López, S. Balme, J.M. Theis, P. Lebourg, H. Fernandes, A. Neto, A. Duarte, F. Oliveira, F. Reis, K. Purahoo, K Thomsen, W. Schiller and J. Kadlecsik
Título: Poster, “PAPI based federation as a test-bed for a common security infrastructure in EFDA sites"
Evento: 6º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research 4 - 8 June 2007 , Inuyama, Japan

Autores: R. Castro, J. Vega, A. Pereira, A. Portas
Título: Poster, “Data distribution architecture based on standard real time protocol"
Evento: 25th Symposium on Fusion Technology, Rostock (Germany), 15-19 September 2008

Autores: R. Castro, K. Kneupner, J. Vega, G. De Arcas, J.M. López, K. Purahoo, A. Murari, A. Fonseca, A. Pereira, A. Portas and JET-EFDA Contributors
Título: Poster, “Real-Time Remote Diagnostic Monitoring test-bed in JET"
Evento: 7º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 15 - 19 June 2009 , Aix-en-Provence , France

Autores: R. Castro, J. Vega, T. Fredian, K. Purahoo, A. Pereira, A. Portas
Título: Poster, “Securing MDSplus in a multi-organization environment"
Evento: 7º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 15 - 19 June 2009 , Aix-en-Provence , France

Bibliografía

[1] R. Castro, D. R. Lopez, The PAPI system: point of access to providers of information, Computer Networks, Volume 37, Issue 6, (2001) 703-710

[2] R. Castro, D. López. "PAPI, una propuesta de RedIRIS para el acceso ubicuo a recursos de información". El Profesional de la Información, Vol 10, Noviembre 2001, Pages 11-14, ISSN: 1386-6710

[3] R. Castro, D.R. López and J. Vega. "An authentication and authorization infrastructure: The PAPI system", Fusion Engineering and Design, Volume 81, Issues 15-17, July 2006, Pages 2057-2061

[4] D. Kristol, L. Montulli. "HTTP State Management Mechanism". RFC 2965, October 2000.

[5] Sciberras, A., Ed., "Lightweight Directory Access Protocol (LDAP): Schema for User Applications", RFC 4519, June 2006

[6] Paul N. Weinberg , James R. Groff . "SQL (The Complete Reference)". McGraw-Hill Osborne Media (2002)

[7] J. Hassell. "RADIUS". O'Reilly Media, Inc. (2002)

[8] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[9] H. Yadava. "The Berkeley DB book". Apress. (2007)

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. " Hypertext Transfer Protocol -- HTTP/1.1". RFC 2616, June 1999

[11] Joan Daemen and Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002. ISBN 3-540-42580-2.

[12] J. Kahan, "W3C Libwww Documentation". World Wide Web Consortium. "http://www.w3.org/Library/User/".

[13] J. Vega, C. Crémy, E. Sánchez, A. Portas. “The TJ-II data acquisition system: an overview”. Fusion Engineering and Design, 43 (1999) 309-319

[14] C. Crémy, J. Vega, C. M. Dulya, E. Sánchez, A. Portas. “Multiprocessor architecture to handle TJ-II VXI-based digitization channels”. Review of Scientific Instruments, 70, 1 (1999) 513-516

[15] C. M. Dulya, C. Crémy, J. Vega, E. Sánchez, A. Portas. “Applying object oriented concepts to online data acquisition”. Review of Scientific Instruments, 70, 1 (1999) 517-520

[16] E. Sánchez, A. Portas, J. Vega, J. M. Agudo, K. J. McCarthy, M. Ruiz, E. Barrera, S. López. “Autonomous acquisition systems for TJ-II: controlling instrumentation with a 4th Generation Language”. Fusion Engineering and Design. 71 (2004) 123-127

[17] J. Vega, E. Sánchez, C. Crémy, A. Portas, C. M. Dulya, J. Nilsson. “TJ-II data retrieving by means of a client/server model”. Review of Scientific Instruments, Vol. 70, No. 1, Pag. 498, Ene. 1999

[18] E. Sñanchez, J. Vega, C. Crémy, A. Portas. "Accessing TJ-II data with remote procedure call". Review of Scientific Instruments 72, 1 (2001) 525-529

[19] E. Sánchez, A. Portas, J. Vega "A relational database for physical data from TJ-II discharges". Fusion Engineering and Design 60 (2002) 341-346

[20] J. Vega, C. Crémy, E. Sánchez, A. Portas, J. A. Fábregas y R. Herrera. “Data management in the TJ-II multilayer database”. Fusion Engineering and Design. 48 (2000) 69-75

[21] J. Vega, C. Crémy, E. Sánchez, A. Portas, S. Dormido. “Encoding technique for a high data compaction in data bases of fusion devices”. Review of Scientific Instruments. 67, 12 (1996) 4154-4160

[22] J. Vega, E. Sánchez, A. Portas, M. Ochando, A. Mollinedo, J. Muñoz, M. Ruiz, E. Barrera, S. López. “A distributed synchronization system for the TJ-II local area network”. Fusion Engineering and Design, 71 (2004) 117-121

[23] J. Vega, E. Sánchez, A. Portas, A. Pereira, A. Mollinedo, J. A. Muñoz, M. Ruiz, E. Barrera, S. López, D. Machón, R.Castro, D. López. “Overview of the TJ-II remote participation system”. Fusion Engineering and Design. 81 (2006) 2045-2050

[24] J. Vega, E. Sánchez, A. Portas, A. Pereira, A. López, E. Ascasíbar, S. Balme, Y. Buravand, P. Lebourg, J. M. Theis, N. Utzel, M. Ruiz, E. Barrera, S. López, D. Machón, R. Castro, D. López, A. Mollinedo, J. A. Muñoz. “TJ-II operation tracking from Cadarache”. Fusion Science and Technology. 50 (2006) 464-471

[25] Liu Yan. " DataSocket Technology and Its Application in Remote Data Transmission Measurement". Electronic Measurement and Instruments, 2007. ICEMI apos;07. 8th International Conference on Volume , Issue , Aug. 16 2007-July 18 2007 Page(s):2-292 - 2-295

[26] J. Vega, E. Sánchez, A. Portas, M. Ruiz, E. Barrera, S. López. “A multi-tier approach for data acquisition programming in the TJ-II remote participation system”. Review of Scientific Instruments. 75, 10 (2004) 4251-4253

[27] "Jakarta Commons HTTPClient". http://hc.apache.org/httpclient-3.x/

[28] R. Tschalär. "HTTPClient Library". http://www.innovation.ch/java/HTTPClient/

[29] M. Marinilli. "Java Deployment with JNLP and WebStart". Addison Wesley. September 2001. ISBN: 0672321823

[30] Jason Brittain, Ian F. Darwin. "Tomcat: The Definitive Guide". O'Reilly Media. June 2003

[31] C. Rigney, S. Willens, A. Rubens, W. Simpson. "Remote Authentication Dial In User Service (RADIUS)". RFC 2865 (2000)

[32] C. Rigney. "RADIUS Accounting". RFC 2866 (2000)

[33] R. Castro. "Avanzando en la seguridad de las redes WIFI". Boletín de la red nacional I+D, RedIRIS, Vol 73, Septiembre 2005, Pages 23-32, ISSN: 1139-207X

[34] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Ed. "Extensible Authentication Protocol (EAP)". RFC 3748 (2004)

[35] B. Aboba, D. Simon, P. Eronen. "Extensible Authentication Protocol (EAP) Key Management Framework". RFC 5247 (2008)

[36] Gabriel Lopez, Oscar Canovas, Antonio F. Gomez-Skarmeta, Manuel Sanchez, "A proposal for extending the eduroam infrastructure with authorization mechanisms". Computer Standards & Interfaces, Volume 30, Issue 6, Special Issue: State of standards in the information systems security area, August 2008, Pages 418-423

[37] B. Clifford Neuman, Theodore Ts'o. "Kerberos: An Authentication Service for Computer Networks", IEEE Communications, 32(9):33-38. September 1994

[38] C. Adams, S. Lloyd. " Understanding PKI: Concepts, Standards, and Deployment Considerations". Addison-Wesley. (2002)

[39] "Information technology - Syntactic metalanguage - Extended BNF". ISO/IEC 14977 (1996)

[40] E. Rescorla. "HTTS: HTTP over SSL". RFC 2818 (2000)

[41] R. Castro, J. Vega, A. Portas, D. R. López, et al. “PAPI based federation as a test-bed for a common security infrastructure in EFDA sites”. Fusion Engineering and Design. Volume 83, April 2008, Pages 486-490, ISSN: 0920-3796

[42] D. Winer. "XML-RPC Specification". http://www.xmlrpc.com/spec (2003)

[43] A. Neto, H. Fernandes, D. Alves, D.F. Valcarcel, et al. "A standard data access layer for fusion devices R&D programs", Fusion Engineering and Design, Volume 82, Issues 5-14, Proceedings of the 24th Symposium on Fusion Technology - SOFT-24, October 2007, Pages 1315-1320

[44] "OpenID Specifications". http://openid.net/developers/specs/

[45] Mark Needleman. "The Shibboleth Authentication/Authorization System". Serials Review, Volume 30, Issue 3, 2004, Pages 252-253

[46] E. Maler, P. Mishra, R. Philpott, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML)v1.1", OASIS Standard, September 2003

[47] Toshiro Takase, Naohiko Uramoto, “XML Digital Signature System Independent of Existing Applications”, IEEE Proceedings of the 2002 Symposium on Applicaions and the Internet (SAINT’02w)

[48] R. Castro, J. Vega, A. Portas, A. Pereira, et al. "EFDA-Fed: European federation among fusion energy research laboratories" Journal: Campus-Wide Information Systems, Volume 25, 2008, Pages: 359 - 373, ISSN: 1065-0741

[49] "XWiki technical documentation". http://platform.xwiki.org/xwiki/bin/view/Main/Documentation (2009)

[50] G.Wilkins, J. Caristi, "Jetty 6 Architecture". http://docs.codehaus.org/display/JETTY/Architecture (2009)

[51] "Limesurvey documentation". http://docs.limesurvey.org/

[52] Thomas W. Fredian, Joshua A. Stillerman, "MDSplus. Current developments and future directions", Fusion Engineering and Design, Volume 60

[53] R. Castro, J. Vega, T. Fredian, K. Purahoo, A. Pereira, A. Portas. “Securing MDSplus in a multi-organization environment". Approved for publishing in "Fusion Engineering and Design". 7º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 15 - 19 June 2009 , Aix-en-Provence , France

[54] R. Castro, K. Kneupner, J. Vega, G. De Arcas, et al. “Real-Time Remote Diagnostic Monitoring test-bed in JET". Approved for publishing in "Fusion Engineering and Design". 7º IAEA Technical Meeting on Control, Data Acquisition, and Remote Participation for Fusion Research, 15 - 19 June 2009 , Aix-en-Provence , France

[55] S. Tilak, P. Hubbard, M. Miller, T. Fountain. "The Ring Buffer Network Bus (RBNB) DataTurbine Streaming Data Middleware for Environmental Observing Systems". e-Science and Grid Computing, IEEE International Conference on. Volume , Issue , 10-13 Dec. 2007 Page(s):125 - 133