MiddleWare y Applet

Análisis de arquitectura y solución de firma digital del documento de identidad electrónico (eID) de Bélgica, y definición de los cambios requeridos para adaptar las soluciones de eID Middleware y eID Applet al SNCD de Costa Rica

Rolosa HyJ S.A. - MICITT

20 de Enero de 2014

Resumen

El presente documento presenta un análisis de la arquitectura actual de las soluciones de firma digital del documento de identidad electrónico (eID) de Bélgica, del Middleware y Applet. Asimismo presenta una definición de los cambios requeridos para adaptar las soluciones de eID Middleware y eID Applet al SNCD de Costa Rica.

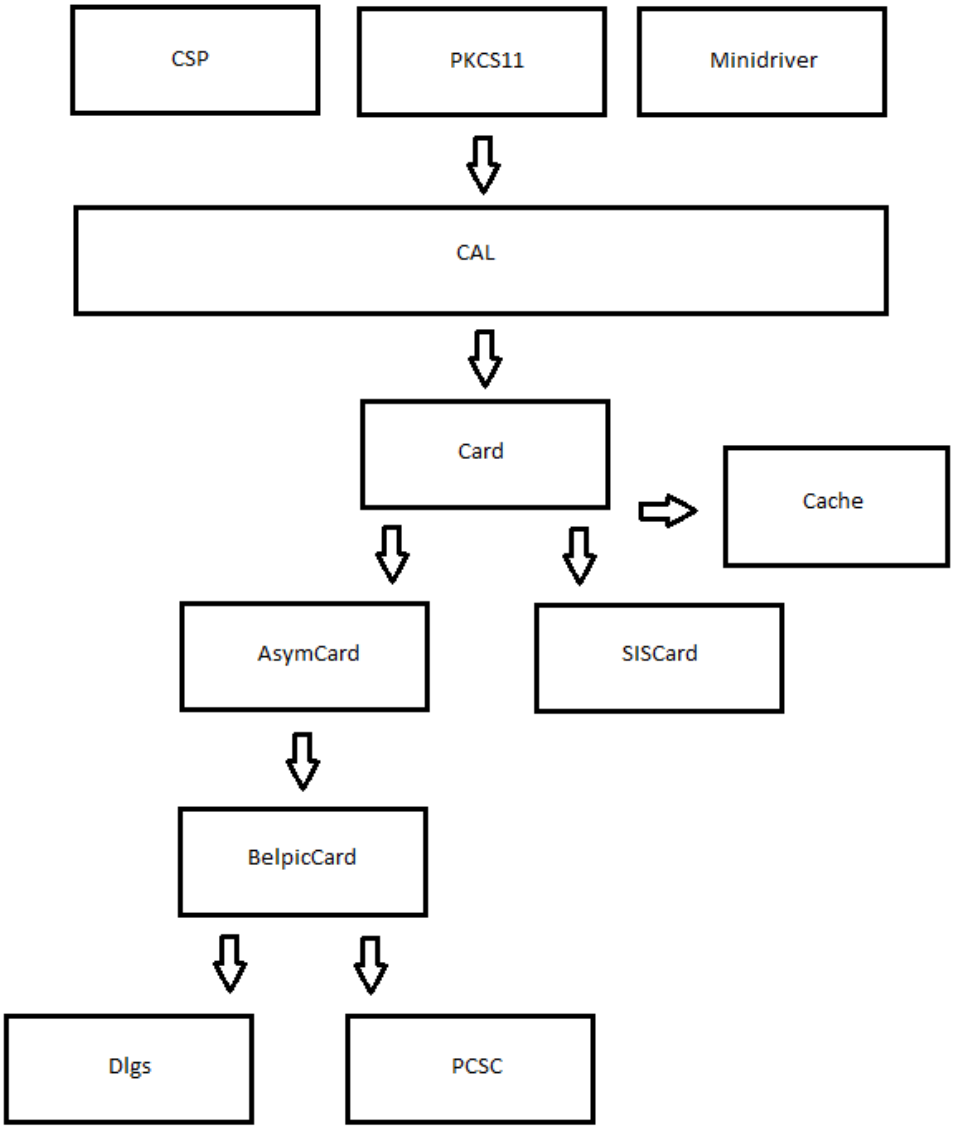
Tabla de contenido

Middleware

El proyecto del eID Middleware de Bélgica ha sido desarrollado bajo los lenguajes C/C++, persiguiendo brindar soporte nativo a las tarjetas eID de Bélgica.

Es en este sentido que la implementación ha llegado a la interacción con las tarjetas de Bélgica a nivel de comandos APDU.

Esta es la estructura general por bloques del eID Middleware se muestra a continuación:



En el grafico anterior se puede observar la dependencia dura que existe entre los módulos PKCS11, CSP y Minidriver con el CAL (Card Abstraction Layer - Capa de Abstracción y acceso físico a la tarjeta).

A continuación se presenta un ejemplo especifico de la dependencia dura que existe entre el módulo PKCS11 con el CAL (Card Abstraction Layer).

Procedimiento básico de inicialización del módulo PKCS11

1. En el archivo `mw/pkcs11/src/general.c:95` se tiene la siguiente llamada a función:

```
C_Initialize()    ->   cal_init();
```

2. Luego en la línea 211 `mw/pkcs11/src/general.c:211` se puede obtener la lista de Card Readers:

```
C_GetSlotList()  -> cal_token_present()  ->  cal_update_token()  ->
CReader::Status()  -> Connect() -> CardConnect()
```

3. En el archivo `mw/cardlayer/src/cardFactory.cpp: 95` se realiza la conexión con la tarjeta inteligente.
   Esta conexión busca una tarjeta conocida de acuerdo al driver implementado dentro del cardlayer o como plugin:

```
CardConnect()   ->
GetCardInstance()                cardFactory.cpp: 138
BeidCardGetInstance()          cardFactory.cpp: 143
SISPluginReadData                cardFactory.cpp: 146
```

4. La definición de las tarjetas conocidas se encuentra en:

```
mw/cardlayer/src/cardpluginbeid
mw/cardlayer/src/cardpluginsis
mw/cardlayer/src/cardpluginsis_acr38u
```

5. La implementación de los drivers para las tarjetas conocidas, como se espera es a bajo nivel, siguiendo ISO7816 y PKCS#15, esto puede ser observado en el archivo
   `mw/pkcs11/src/session.c:340`

```
C_Login()  ->  cal_logon()  -> CReader::PinCmd()  -> CCard::PinCmd  ->  Beidcard::PinCmd()
```

```
.....
oResp = SendAPDU(oAPDU);
.....
unsigned long ulSW12 = getSW12(oResp);
if (ulSW12 == 0x9000)
    bRet = true;
else if (ulSW12 == 0x6983)
    ulRemaining = 0;
else if (ulSW12 / 16 == 0x63C)
    ulRemaining = ulSW12 % 16;
.....
```

Esta dependencia restringe cualquier modificación que se desee realizar para que el Middleware pueda reconocer tarjetas diferentes a las utilizadas en Bélgica, puesto que se deberían conocer datos físicos de las nuevas tarjetas para poder realizar la estructuración de comandos APDU que permitan leer datos de estas nuevas tarjetas.

Los datos físicos para la creación de un driver o plugin para la tarjeta de Firma Digital de Costa Rica, que pueda ser utilizado dentro del eID Middleware, no son de dominio público, por lo cual este enfoque de adaptación no es viable.

Puesto que es necesario romper la dependencia que existe con el módulo CAL para poder utilizar con el eID Middleware cualquier otra tarjeta diferente a las de Bélgica, los cambios de adaptación requerirán de ser realizados en el módulo PKCS11

Módulo CSP

La implementación del Crypto-Service-Provider del Middleware esta ligada físicamente a las tarjetas inteligentes de Bélgica. Este proyecto utiliza el CAL (Card Layer) para la lectura de la información de las tarjetas inteligentes. No es posible realizar cambios sobre el CSP para adaptarlo a la utilización de las tarjetas inteligentes de Firma Digital de Costa Rica sin contar con la información de acceso físico a dichas tarjetas, esta información comprende básicamente detalles del driver de las mismas y comandos APDU.

Módulo Tokend

El Tokend para Mac es equivalente a un CSP en Windows. Al igual que el módulo CSP, la implementación del Tokend del Middleware esta ligada físicamente a las tarjetas inteligentes de Bélgica por medio del Card Layer.

Módulo Certprop

El módulo Certprop tiene como objetivo el registro de los certificados digitales dentro de "Windows Certificate Store" . Al igual que el módulo CSP, la implementación del Certprop del Middleware esta ligada físicamente a las tarjetas inteligentes de Bélgica por medio del Card Layer.

Cambios requeridos para la adaptación del Middleware

Los principales cambios requeridos para la adaptación de la solución eID Middleware para el SNDC de Costa Rica se listan a continuación por orden de relevancia:

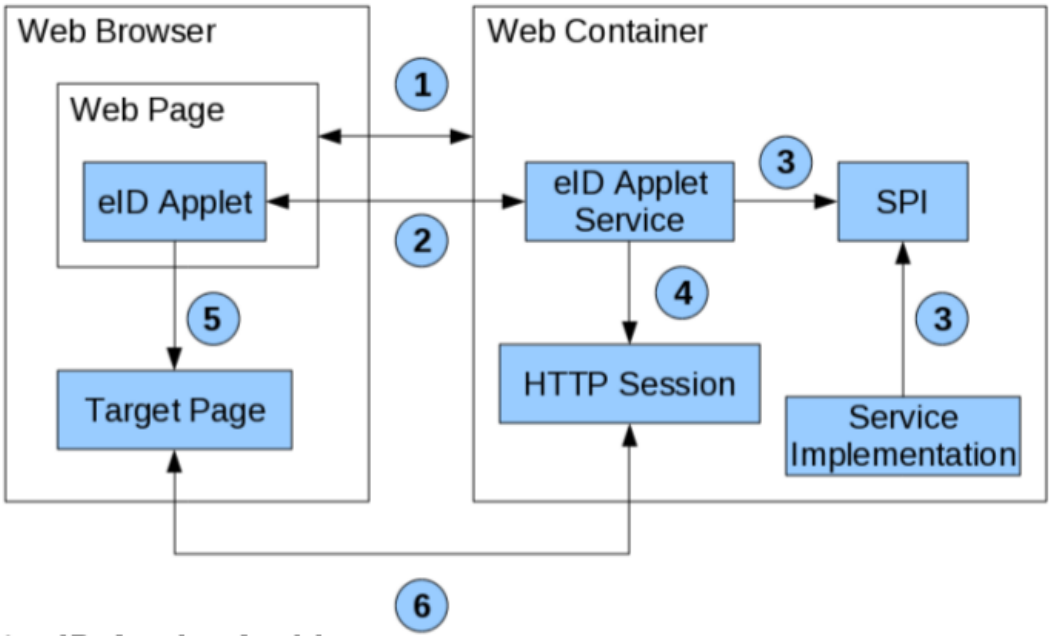
I. Inclusión de la librería PKCS#11 de las tarjetas inteligentes de Firma Digital de Costa Rica creada por Athena, dentro del proyecto pkcs11 del Middleware.
   La forma de inclusión se muestra a continuación:

```
CK_RV C_Login(CK_SESSION_HANDLE hSession,  /* the session's handle */
         CK_USER_TYPE      userType,  /* the user type */
         CK_CHAR_PTR       pPin,      /* the user's PIN */
         CK_ULONG          ulPinLen)  /* the length of the PIN */
{
      return   HANDLER_LIBRERIA_ATHENA_FIRMA_DIGITAL->C_Login(hSession, userType, pPin,
ulPinLen);    //llamada a la funcion C_Login implementada en la libreria de Athena
}
```

Applet

El proyecto del eID Applet de Bélgica ha sido desarrollado siguiendo una arquitectura orientada a servicios.

La arquitectura y el detalle del proceso de la información se muestran a continuación:



En el paso (1) el **Web Browser** (navegador web) carga la pagina web que contiene la referencia al Applet. El navegador web luego continua con la carga del Applet a través del plugin JRE (Java Runtime Enviroment). Una vez que el Applet ha sido completamente cargado, este mismo inicia el protocolo con el **Applet Service** (Servicio para el Applet que se soporta en el servidor) como se muestra en el paso (2). Para las operaciones requeridas como la Autenticación del Titular de algún Certificado Digital o la firma digital de documentos electrónicos, se requiere configurar los componentes necesarios en el **SPI** (Proveedor de Servicios) los cuales son llamados en el paso (3) desde el Applet Service mientras se continua con la ejecución del protocolo. Una vez terminada la ejecución del protocolo, en el paso (4) , el Applet Service envía los atributos necesarios al contexto **HTTP Session** (Sesion HTTP). Finalmente en el paso (5) el Applet hace que el navegador web se direccione a la **Target Page** (Pagina de destino), la cual puede ya acceder a los atributos de que se hicieron disponibles en el contexto HTTP Session, paso (6), y que fueron puestos a disponibilidad por el Applet Service.

El proyecto de eID Applet, ha sido creado y administrado mediante Apache Maven 3. Los subproyectos mas importantes que incluye son:

1. **eID Applet Shared**

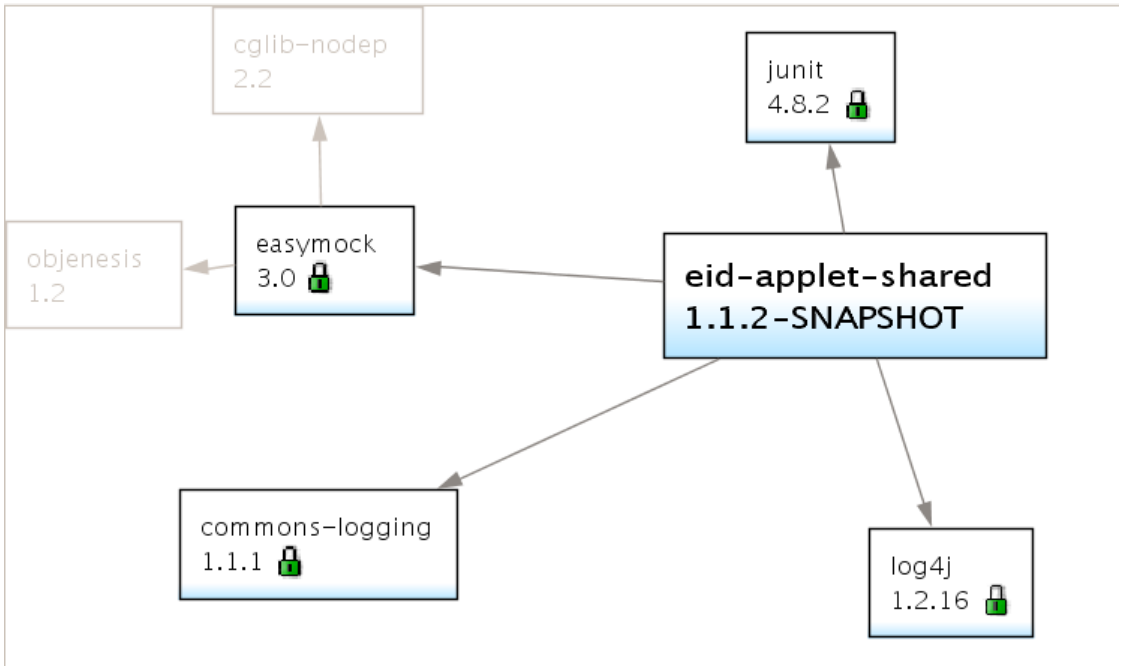
   Este proyecto contienen todas las clases que exponen funcionalidad a los demás proyectos dentro del proyecto principal eID Applet.

   El package **be.fedict.eid.applet.shared** define todas las clases para los mensajes a intercambiar entre el Applet y el Applet Service

   El package **be.fedict.eid.applet.shared.annotation** define todas la anotaciones que han sido especificadas específicamente para los fields que son incluidos en los mensajes parte del protocolo de comunicación entre el Applet y el Applet Service.

   El package **be.fedict.eid.applet.shared.protocol** define todas las interfaces y lógica de transporte para la comunicación entre el Applet y el Applet Service.

   El siguiente gráfico muestra todas las dependencias que este proyecto tiene con librerías externas como las define Apache Maven:



2. eID Applet Core

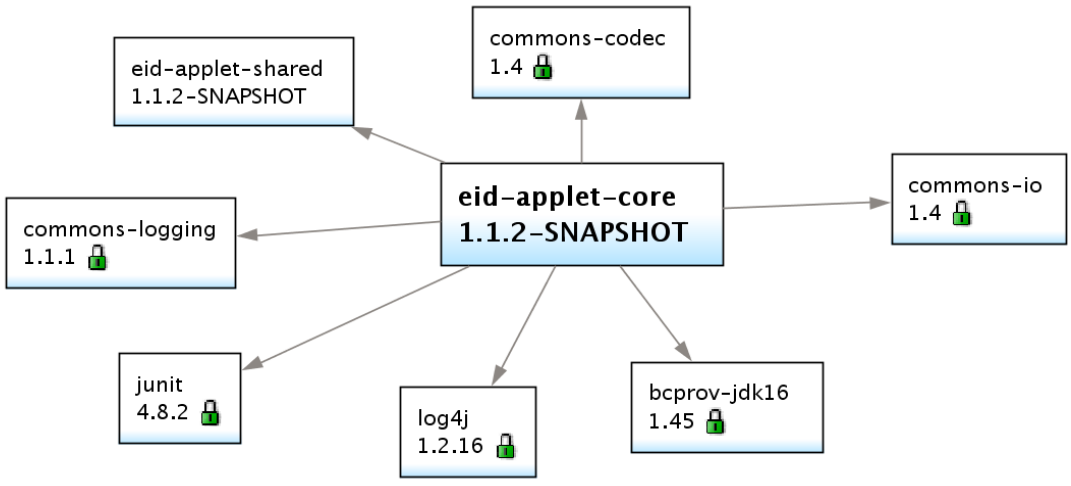
Este proyecto contiene todas las clases que definen al mismo Applet desde la Interfaz de Usuario como las capas de acceso a las tarjetas inteligentes.

El package **be.fedict.eid.applet** define la Interfaz de usuario que el Applet contiene así como toda la funcionalidad del componente. La clase **Controller** es el punto de interacción entre el Applet mismo como componente grafico y la funcionalidad de acceso a la tarjeta inteligente.

El package **be.fedict.eid.applet.sc** define el acceso físico a las tarjetas inteligentes. Las clases implementadas en este package han sido diseñadas para utilizar el soporte PC/SC que Java ha hecho disponible en su package **javax.smartcardio**. Estas clases implementan la comunicación física por medio de comandos APDU. Estos comandos solamente funcionan con las tarjetas eID de Bélgica y no son compatibles en ningún nivel con las tarjetas de Firma Digital de Costa Rica.

El package **be.fedict.eid.applet.io** define las clases para el procesamiento de Http Urls para asegurar la integridad de la ejecución del Applet en entornos seguros de SSL.

El siguiente gráfico muestra todas las dependencias que este proyecto tiene con librerías externas como las define Apache Maven:

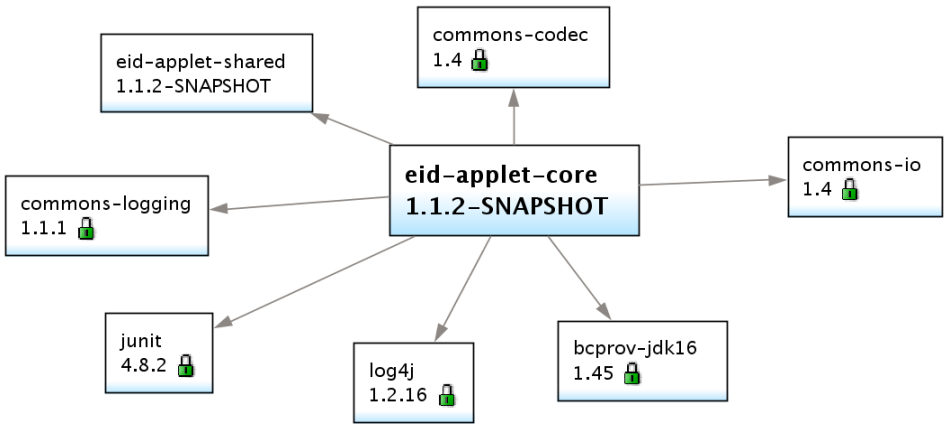


3. **eID Applet Package**

Este proyecto se encarga del empaquetado del Applet para la obtención del archivo JAR final que es el que se incrusta en las aplicaciones web del lado del cliente y que el navegador web cargará.

El empaquetado se hace de las clases del proyecto **eid-applet-shared** y **eid-applet-core.**

El siguiente gráfico muestra todas las dependencias que este proyecto tiene con librerías externas como las define Apache Maven:



4. **eID Applet JavaScript**

Este proyecto mantiene una copia local del código Javascript del "Deployer" (deployJava.js) que se utiliza ampliamente para la incrustación de Java Applets en las aplicaciones web.

Este proyecto no tiene ninguna dependencia que defina Apapche Maven.

5. **eID Applet Service SPI**

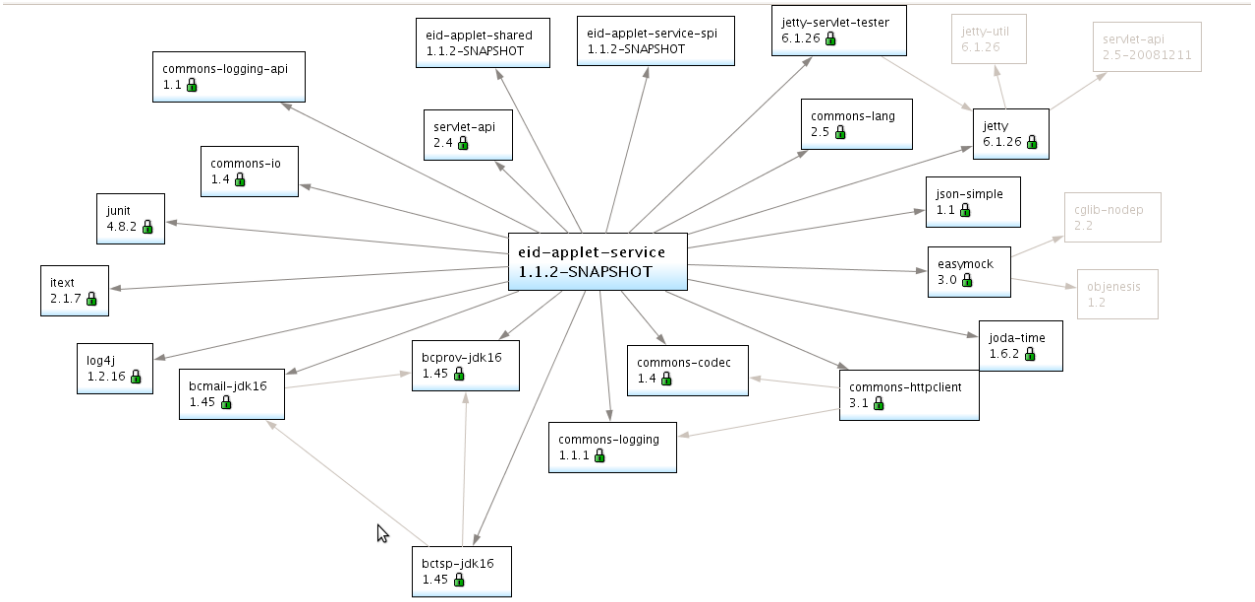
Este proyecto (**Proveedor de Servicios**) contiene todas las interfaces para la implementación de un servicio especifico que se hará disponible en el lado del servidor. Cualquier componente-servicio (EJB3) session-bean tiene que ser registrado en algún lugar en JNDI y la ubicación JNDI tiene que ser informada para que el servlet correspondiente pueda consumir el servicio expuesto por el session-bean.

Este proyecto no tiene ninguna dependencia que defina Apapche Maven.

6. eID Applet Service

Este proyecto contiene el Applet Service Servlet. Este servlet tiene que ser utilizado por las aplicaciones web para asegurar la comunicación entre le Java EE servlet container y el eID Applet. Este servlet proveerá de atributos dentro del componente session HTTP luego de una identificación exitosa del navegador web a través del eID Applet.

El siguiente gráfico muestra todas las dependencias que este proyecto tiene con librerías externas como las define Apache Maven:



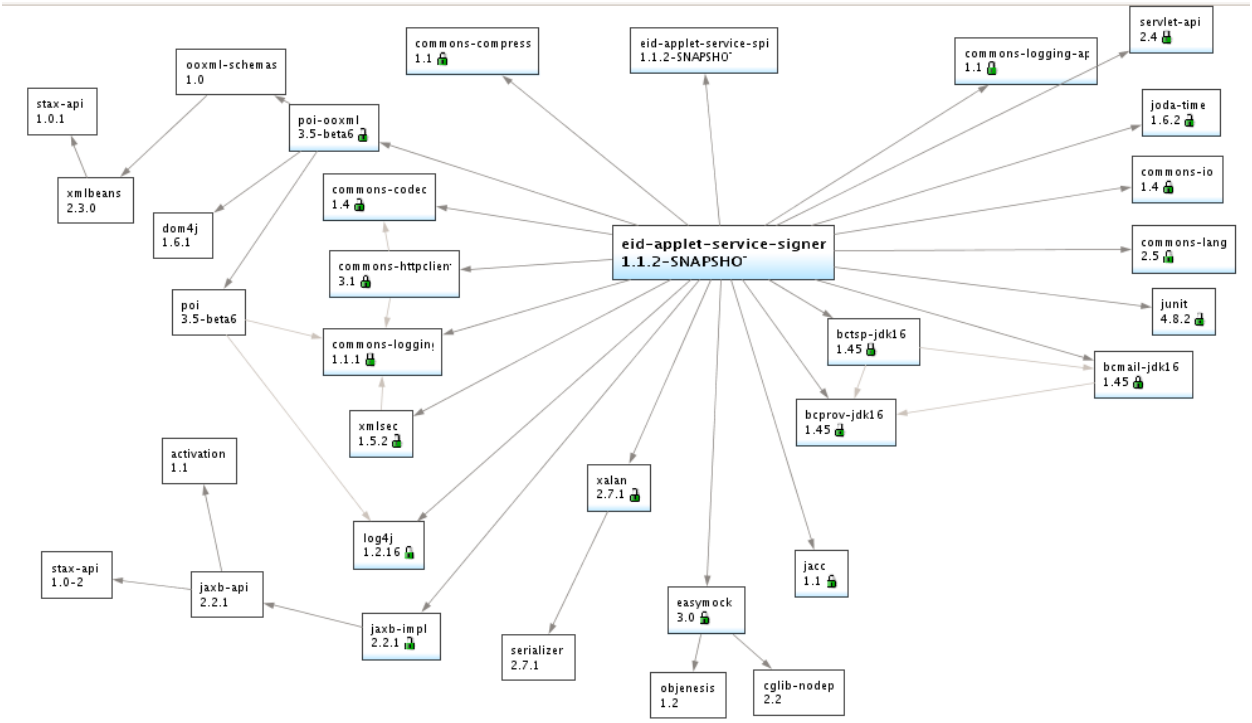
7. eID Applet Service Signer

Este proyecto proveen diversas implementaciones base del SignatureService SPI

Las implementaciones de servicio de firma más importantes provistas por el Applet son:

- Firmas ODF 1.2 (Soportadas por OpenOffice.org 3.1/3.2)
- Office OpenXML 9Soportadas por Microsoft Office 2007/2010
- Firmas CMS (PKCS#7)

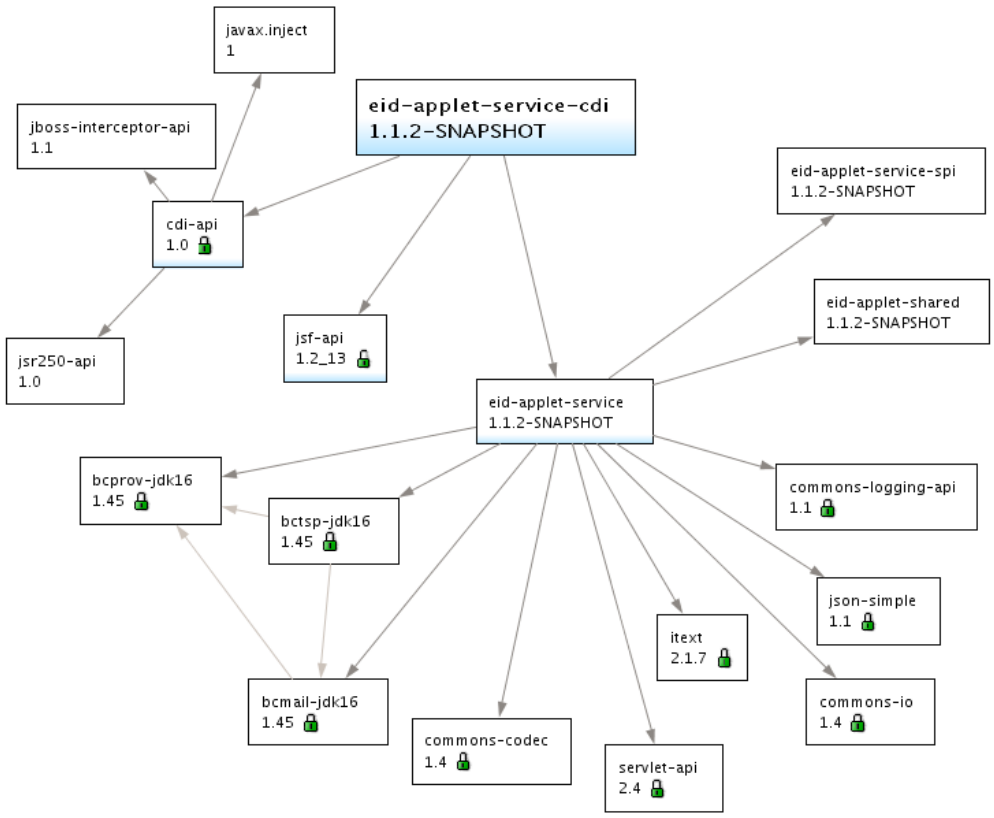
El siguiente gráfico muestra todas las dependencias que este proyecto tiene con librerías externas como las define Apache Maven:



8. eID Applet Service CDI

Este proyecto contiene anotaciones para la utilización de "Context and Dependency Injection" para aplicaciones Java Server Faces 2.0.

El siguiente gráfico muestra todas las dependencias que este proyecto tiene con librerías externas como las define Apache Maven:



9. eID Applet SDK

Este proyecto permite la generación del SDK correspondiente a la versión en la que se encuentra el proyecto principal. Realiza una compilación de los proyectos anteriormente listados y recolecta los archivos JAR resultantes dentro de un solo paquete. Para este proceso es necesario utilizar un **eToken** que contenga el certificado digital de firma para firmar el archivo JAR del Applet.

Cambios requeridos para la adaptación del Applet

Los principales cambios requeridos para la adaptación de la solución eID Applet para el SNDC de Costa Rica se listan a continuación por orden de relevancia:

II. Inclusión de soporte PKCS11 dentro del proyecto eid-applet-core.

El package **be.fedict.eid.applet.sc** necesita ser modificado para que el Applet pueda interactuar con la librería resultante de la compilación del Middleware que sigue el estándar PKCS11.
No es posible utilizar el soporte PC/SC que Java ha hecho disponible en su package **javax.smartcardio**, debido a que la comunicación física por medio de comandos APDU y su definición no es de dominio público. Únicamente se dispone de las librerías de acceso a las tarjetas inteligentes de Firma Digital de Costa Rica y estas serán incluidas dentro de la solución resultante de las modificaciones al Middleware.

III. Adecuación del Controlador principal del Applet para la interacción con las modificaciones requeridas por el soporte PKCS11

El package **be.fedict.eid.applet** necesita ser modificado específicamente en la clase Controller, para poder consumir las funcionalidades que la interfaz PKCS11 provee.

IV. Traducción de los mensajes utilizados en la interfaz de usuario que el Applet expone.
En la actualidad únicamente se disponen de mensajes en Ingles, Francés y Alemán.