Ingeniería de Telecomunicación

Escuela Superior de Ingenieros

Universidad de Sevilla

Proyecto Fin de Carrera

Servicio Web para la Identificación de Personas

Pilar María Juan Montilla

Tutora: Isabel Román Martínez

Sevilla, Octubre de 2007

Quisiera dar las gracias muy especialmente a mis padres, Francisco y Pilar, por su apoyo incondicional durante todos estos años de carrera.

A mis amigos y compañeros, por los momentos inolvidables que hemos vivido dentro y fuera de la Escuela.

Y por último, a mis amigas Blanca, Cristina y Teresa, porque su amistad es uno de los grandes tesoros que me llevo de estos años de carrera.

Muchas gracias a todos.

Índice de Contenidos

PARTE I : INTRODUCCIÓN GENERAL

1 Introducción

1.1 Motivación y Objetivos

Las principales motivaciones de este proyecto surgen de la importancia de la identificación de personas y la gestión de datos demográficos. Se pretende disponer de un sistema de identificación de personas con información normalizada y que sea fácilmente accesible a través de la tecnología de servicios web. La idea es contribuir a la solución de la problemática actual que supone la existencia de múltiples sistemas de identificación de personas, cada uno de ellos con un criterio de asignación de identificadores de persona distinto. Para que todos estos sistemas se relacionen e intercambien información demográfica entre sí resulta indispensable que todas las personas estén identificadas de forma unívoca. Uno de los sistemas que fue desarrollado para cumplir estos objetivos es el Servicio de Identificación de Personas (PIDS) de CORBAmed.

Una implementación del servicio PID mencionado fue desarrollado con anterioridad en el proyecto fin de carrera de Mª Ángeles Repullo López "Servidor demográfico basado en normas del CEN y CORBAmed", y fue ampliado y mejorado por Mª José Caminero Marqués para su proyecto "Pasarela Java-XML para GPICS no clínicos y aplicación en un servidor demográfico". En el presente proyecto se parte de este sistema de información y se pretende implementar un servicio web que ofrezca las funcionalidades ya realizadas pero ahora de forma remota.

Más adelante se explica en detalle en qué consiste el Servicio de Identificación de Personas de CORBAmed. En líneas generales se pueden resaltar dos funciones del mismo:

- Identificar de forma unívoca a una persona dentro de un colectivo a partir de ciertas características distintivas de la misma.

- Extraer información demográfica y administrativa de una persona partiendo de su identificador.

Para probar el correcto funcionamiento del servicio web implementado se ha desarrollado también una interfaz web capaz de invocar el servicio pasándole los

parámetros necesarios.

1.2 Fases de la Implementación

En el desarrollo del proyecto se pueden distinguir una serie fases que se comentan a continuación:

1. Documentación: Lo primero en cualquier proyecto es documentarse sobre la materia con la que se va trabajar. En esta fase se profundizó en la especificación del Servicio de Identificación de Personas (PIDS) de CORBAmed, así como en los distintos estándares HealthInformatics-Data Types y HealthInformatics- prEN 14822-2:2003 del CEN (*Comité Europeo de Normalización*). Con el estudio de proyectos fin de carrera anteriores se agilizó mucho esta labor. Posteriormente, se hizo un estudio de los servicios web: qué son, ventajas que aportan, las distintas tecnologías y herramientas existentes para el desarrollo de los mismos,etc.

2. Instalación del entorno de desarrollo del proyecto, el cual se detallará más adelante. En el apartado 6 de la memoria se comentan las distintas herramientas y tecnologías utilizadas para el desarrollo del proyecto.

3. Reutilización de código: Una vez que nos habíamos documentado debidamente, se habían estudiado varios proyectos fin de carrera anteriores, y se tenía instalado el entorno de trabajo, el siguiente paso fue la modificación del código del servicio PIDS para adaptarlo a nuestras necesidades, así como la base de datos de partida.

4. Diseño y creación del código java que implementa el servicio web del PIDS. Aquí se incluye tanto el código del servidor como el del cliente.

5. Desarrollo de una interfaz web para validar el correcto funcionamiento del servicio web y facilitar su uso.

6. Redacción de esta memoria.

En la siguiente figura se presenta mediante un diagrama de Gantt las fases de implementación comentadas anteriormente:

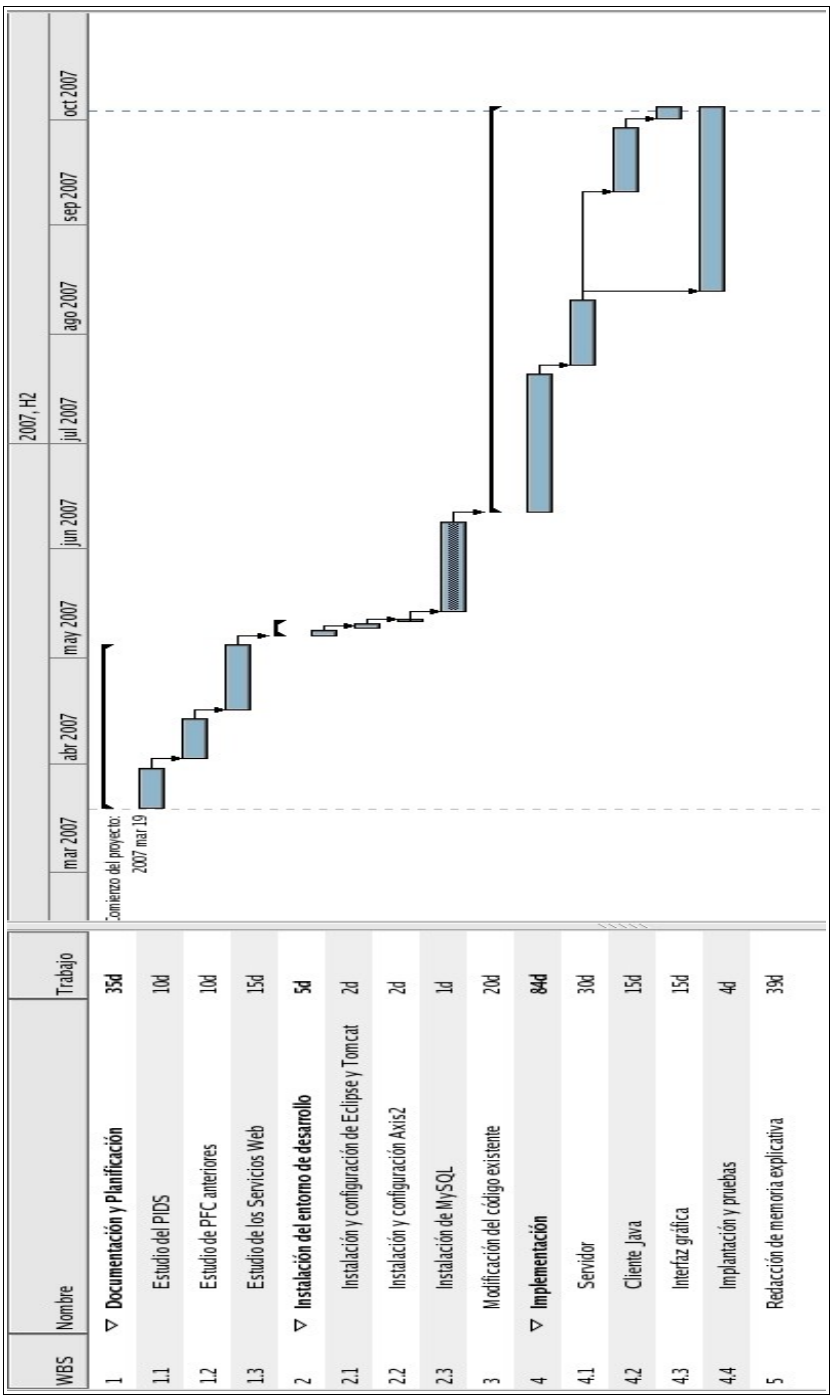

Figura 1.1: Diagrama de Gantt

1.3 Material y Método

Para la realización del servicio web se han usado una serie de programas para las tareas de desarrollo, despliegue de la aplicación, almacenamiento de los datos y pruebas.

Primeramente, indicar que la máquina usada para el desarrollo de la aplicación ha sido un portátil con procesador Intel Core 2 Dúo, con 2 GB de RAM y con sistema operativo Linux Fedora Core 6 (instalado en la máquina virtual Vmware Workstation 5.5.3 sobre Windows XP).

La herramienta de desarrollo usada ha sido Eclipse 3.2.1, la cual facilitó mucho la creación y depuración del código generado, así como la realización de pruebas.

Se ha usado una base de datos MySQL, por ser un sistema administrador de base de datos "open source" (de código abierto) de los más usados en entornos UNIX y por ser muy sencillo y fácil de usar. Adicionalmente se instaló phpMyAdmin para agilizar el manejo y la gestión de la base de datos.

Por último, se ha utilizado el contenedor de aplicaciones Apache Tomcat 5.5.20, sobre el cual se he desplegado Axis2 1.2. Axis2 es el proyecto de Apache que nos facilita el desarrollo de aplicaciones con servicios web. Más adelante se explicará más específicamente qué es, para qué sirve y qué facilidades nos proporciona a la hora de trabajar con servicios web.

PARTE II : INTRODUCCIÓN TEÓRICA

2 Servicio de Identificación de Personas de CORBAmed

2.1 CORBA y CORBAmed

CORBA (*Common Object Request Broquer Architecture*) es un estándar desarrollado por el OMG (*Object Management Group*) para la invocación de métodos sobre objetos en entornos heterogéneos. Los entornos heterogéneos son aquellos en los que encontramos una gran variedad de productos hardware y software. CORBA permite la comunicación entre aplicaciones con independencia de su localización o de quién las creó.

CORBA define su propio modelo de objetos, basado en la definición de las interfaces de los objetos mediante el lenguaje IDL (*Interface Definition Language*), el cual posee un alto nivel de abstracción, lo que le hace independiente del entorno de desarrollo y de la plataforma. La interfaz define qué servicios ofrece el objeto, cómo invocarlos y su implementación. En definitiva, podemos decir que CORBA ha logrado parte de su éxito gracias a la clara separación entre la interfaz y el objeto, así como por seguir una metodología concreta y fácil de seguir.

CORBAmed es el grupo de trabajo que el OMG ha creado para el uso de CORBA en el sector sanitario. La principal misión de CORBAmed es facilitar el acceso a todo tipo de información clínica, para lo cual:

- Promueve la interoperatividad entre diferentes sistemas sanitarios, pertenecientes incluso a distintas organizaciones.

- Garantiza una mayor seguridad y confidencialidad en el intercambio de datos médicos entre organizaciones.

- Define interfaces estandarizadas de servicios de interés en el dominio sanitario.

- Mejora la calidad de la atención sanitaria y reduce costes por medio del uso de tecnología CORBA.

2.2 Necesidad del Servicio de Identificación de Personas

Una persona, a lo largo de su vida, puede llegar a recibir cuidados médicos suministrados por diferentes profesionales sanitarios, y los distintos proveedores sanitarios suelen asignar de forma independiente identificadores (IDs) a sus pacientes.

Según este patrón cada organización asigna IDs que, de forma unívoca, identifican a pacientes dentro de su dominio local de valores de ID. Fuera de ese sistema u organización dichos IDs carecen de significado. Esta forma de manejar los IDs cubre las necesidades de almacenar y recuperar información dentro de la organización local, pero no soluciona la recopilación de información de un paciente procedente de algún sistema externo que utilice distintos IDs. Esto dificulta el acceso al historial clínico completo del paciente. Es por ello que a actual tendencia es la migración hacia sistemas basados en la compartición de datos.

Para identificar a una persona, hay gran variedad de información a usar como:

- Datos demográficos (dirección, lugar de nacimiento, etc).

- Datos administrativos (número de la seguridad social, número de la licencia de conducción, número del DNI, etc).

Hay que tener en cuenta que la información más apropiada para identificar a una persona se compone de aquellos atributos cuyo valor permanece constante o cambia muy lentamente a lo largo del tiempo, ya que podrán almacenarse y usarse posteriormente con fines identificativos.

El PIDS ha sido diseñado para cumplir los siguientes objetivos:

- Soportar de manera simultánea la asignación de IDs dentro de un dominio particular y la correlación de IDs entre múltiples dominios.

- Soportar la búsqueda y localización de personas, tanto en modo interactivo como en no atendido, con independencia del algoritmo utilizado.

- Proteger la confidencialidad de las personas, bajo una amplia variedad de políticas de privacidad y mecanismos de seguridad.

- Definir los niveles apropiados de conformidad para varios grados de sofisticación, desde pequeñas búsquedas en un solo dominio de ID hasta búsquedas sobre varios dominios.

2.3 Modelo de Referencia del PIDS

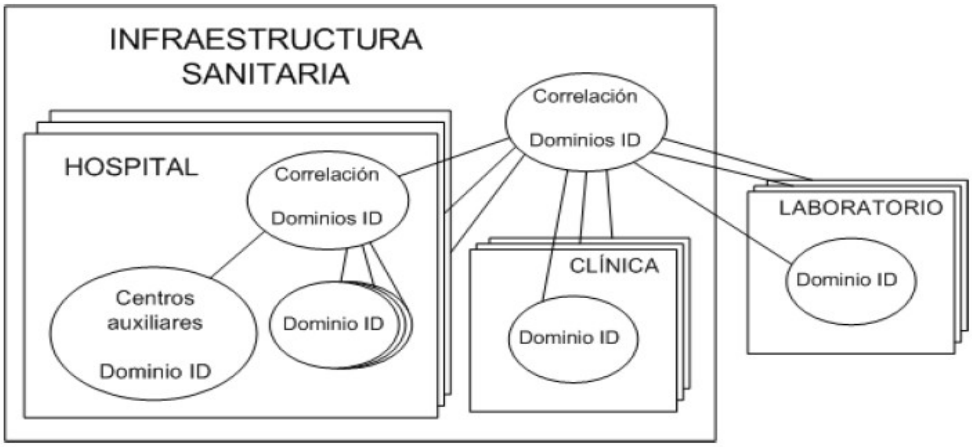

Figura 2.1: Modelo de Referencia del PIDS

En la figura 2.1 se representa el modelo de referencia que proporciona el PIDS, en el que se pueden apreciar los dominios de identificación que normalmente existen en una infraestructura sanitaria.

Un hospital utiliza identificadores pertenecientes a su propio dominio de identificación, trabajando a su vez con otros centros auxiliares pertenecientes al mismo hospital, como pueden ser laboratorios, los cuales tienen también su propio dominio. El hospital es el encargado de establecer correspondencias entre ambos dominios. Una buena infraestructura sanitaria contará con múltiples hospitales, laboratorios, clínicas, cada uno de ellos con su propio dominio de IDs, por lo que una correcta gestión de dominios de IDs es fundamental.

2.4 Modelo de Identificación del PIDS

En la figura 2.2 se han representado los elementos que forman la estructura básica del modelo de identificación del PIDS.

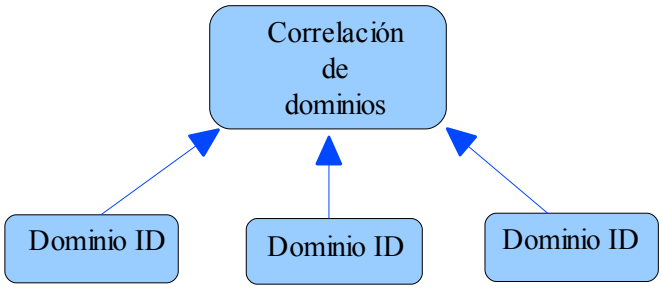

Figura 2.2: Modelo de Identificación del PIDS

El *Dominio ID* (o Dominio de Identificación) es el bloque básico del modelo de identificación del PIDS. En cada dominio se asigna un identificador único para cada persona, por tanto, conociendo el dominio y el identificador podemos tener completamente identificada a una persona. La unidad estructural que coordina el modelo PIDS es la *Correlación de Dominios*, que permite acceder a los perfiles de todas las personas registradas en los Dominios ID participantes.

Idealmente, sólo hay un ID por persona, pero en la realidad puede suceder que a una misma persona se le asigne más de un ID dentro del mismo dominio. Por motivos de consistencia, dentro de un mismo dominio nunca se asignará un mismo ID a más de una persona, ya que resultaría imposible distinguirlas. La unión del identificador de persona y el ID del dominio crea un identificador único para la persona.

En la especificación del PIDS se detallan varias interfaces. Dos de las principales son la interfaz **IdentifyPerson** y la interfaz **ProfileAccess**. La interfaz *IdentifyPerson* básicamente es una consulta usada para localizar a una persona, es decir, encontrar su identificador a partir de algunos de los atributos conocidos acerca de ella. A través de la interfaz *ProfileAccess* se pueden realizar consultas o actualizaciones de los atributos de una persona si se conoce el ID de la misma. Un "profile" o perfil es un conjunto de atributos (nombre y valor) referidos a una persona.

2.5 Diagrama de Herencia del PIDS

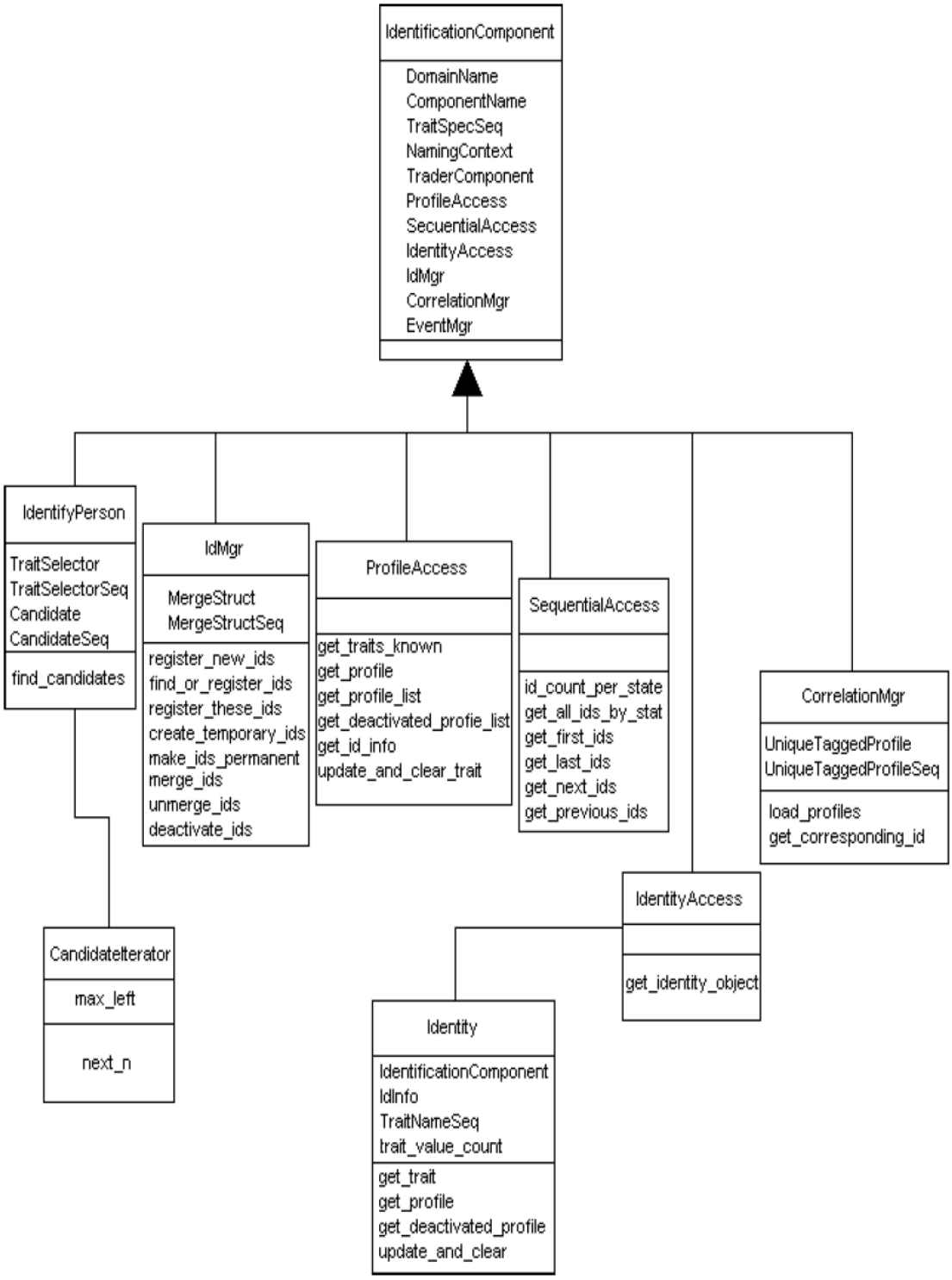

Figura 2.3: Diagrama de Herencia del PIDS

En la figura 2.3 se muestran las múltiples interfaces definidas en el Servicio de Identificación de Personas. Cada interfaz representa una funcionalidad y es opcional, por lo que cada implementación del PIDS implementa sólo las interfaces que necesita.

Todas las interfaces, como se puede observar, heredan de la interfaz **IdentificationComponent**. A su vez, un *IdentificationComponent* tiene un número opcional de interfaces que puede implementar. De esta manera, partiendo de un ID se pueden conocer todas las características disponibles de una persona.

A continuación vamos a ver con más detenimiento las estructuras de los tipos que se utilizan, así como los distintos métodos que se definen en las interfaces *IdentifyPerson* y *ProfileAccess*, ya que son las dos interfaces más relevantes en el desarrollo de este proyecto. De la interfaz ProfileAccess sólo ha sido necesario implementar parte de la interfaz (los métodos *get_traits_known*, *get_profile* y *get_profile_list*).

2.6 Interfaz IdentifyPerson

2.6.1 Principales Tipos usados en la Interfaz

TraitSelector y TraitSelectorSeq

El *TraitSelector* es un parámetro de información que el cliente aporta al servicio para identificar a la persona o personas que se quiere localizar:

```
struct TraitSelector{
      PersonIdService::Trait trait;
      float weight;
};
```

La estructura *TraitSelector* consta de un *Trait* (nombre y valor de un atributo):

```
struct Trait{
      TraitName name;
      TraitValue value;
};
```

El campo *weight* (peso) representa la importancia, que según el cliente, el servidor debe darle a cada *Trait*. No está normalizado el uso que el servidor hará de este peso y por tanto es completamente dependiente de la implementación.

Un *TraitSelectorSeq*, es una secuencia de elementos *TraitSelector*.

Candidate y CandidateSeq

La estructura *Candidate* es devuelta después de hacer una búsqueda de una persona. Contiene el identificador (ID) de la persona, un campo *confidence* (confianza) que indica cómo de bien se ajusta esa persona a los atributos introducidos para hacer la búsqueda, y el conjunto de *Traits* (atributos) pedidos. El uso del parámetro de confianza y cómo se hace su cálculo no está normalizado, y por tanto depende de la implementación. Sin embargo, la norma dice que los campos de peso indicados por el cliente pueden ser utilizados por el servidor para el cálculo de la confianza.

```
struct Candidate{
      PersonId id;
      float confidence;
      PersonIdService::Profile profile;
};
```

Un *CandidateSeq*, es una secuencia de *Candidate*.

Profile

Un *Profile* se define como:

```
Typedef TraitSeq Profile;
```

Siendo *TraitSeq* una secuencia de elementos *Trait*.

SpecifiedTraits

La estructura *SpecifiedTraits* es una secuencia de *TraitSpec*, que son nombres de atributos para los que el servicio debe encontrar su correspondiente valor. Es decir, son los atributos que el cliente quiere conocer sobre la persona buscada.

```
struct TraitSpec{
      TraitName trait;
      boolean mandatory;
      boolean read_only;
      boolean searchable;
};
```

2.6.2 Métodos de la Interfaz

Método find_candidates

```
void find_candidates(
      in TraitSelectorSeq profile_selector,
      in IdStateSeq states_of_interest,
      in float confidence_threshold,
      in unsigned long sequence_max,
      in SpecifiedTraits traits_requested,
      out CandidateSeq returned_sequence,
      out CandidateIterator returned_iterator)
raises(
      TooMany,
      UnknownTraits,
      WrongTraitFormat,
      CannotSearchOn,
      DuplicateTraits,
      InvalidIdState,
      InvalidWeight);
```

Conociendo alguna información que identifique a una persona, un cliente puede pedirle al servidor que le devuelva los candidatos que crea que se corresponden con ese perfil. Sólo deben devolverse aquellos candidatos cuyo estado del identificador se corresponda con los estados de ID válidos introducidos en states_of_interest, que es una

secuencia de estados válidos para los identificadores.

```
enum IdState{UNKNOWN, INVALID, TEMPORARY, PERMANENT, DEACTIVATED}
```

El confidence_threshold, es la confianza mínima exigida para cada candidato devuelto, y sus valores oscilan entre 0.0 y 1.0.

El cliente puede indicar el número máximo de candidatos devueltos mediante sequence_max. Si el servicio encuentra un número mayor de candidatos, se devuelven dentro de un *Iterator*, pudiéndose indicar el tamaño máximo de este mediante iterator_max. El traits_requested es un conjunto de nombres de atributos, cuyos valores debe buscar el servicio para cada candidato devuelto.

Cuando una operación devuelve una secuencia de datos de gran tamaño y excede la capacidad del servidor se genera la excepción *TooMany*. Si el campo searchable de alguno de los SpecifiedTraits tiene valor falso, se lanza la excepción *CannotSearchOn*. Si en alguna secuencia de *Traits* se introduce alguno duplicado se genera la excepción *DuplicateTraits*, si se introduce alguno inválido, se produce una excepción *UnknownTraits* y si el valor del atributo no se corresponde con el tipo esperado se genera la excepción *WrongTraitFormat*. Los campos weight (de profile_selector), pueden tomar valores entre 0.0 y 1.0, y un valor fuera de este rango provocaría una excepción de *InvalidWeight*. Si el estado de algún identificador obtenido como resultado de una operación no coincide con los estados introducidos en *IdStates*, se genera la excepción *InvalidIdState*.

2.7 Interfaz ProfileAccess

2.7.1 Principales Tipos usados en la Interfaz

TaggedProfile y TaggedProfileSeq

La estructura *TaggedProfile* indica un perfil para un ID en particular.

```
struct TaggedProfile{
      PersonId id;
      PersonIdService::Profile profile;
};
```

Un *TaggedProfileSeq* es una secuencia de *TaggedProfile*.

ProfileUpdate y ProfileUpdateSeq

La estructura *ProfileUpdate* indica para un ID qué atributos deben borrarse (del_list) y cuáles modificarse (modify_list) .

```
struct ProfileUpdate{
      PersonId id;
      TraitNameSeq del_list;
      TraitSeq modify_list;
};
```

Un *ProfileUpdateSeq* es una secuencia de *ProfileUpdate*.

2.7.2 Métodos de la Interfaz

Método get_traits_known

```
TraitNameSeq get_traits_known(
      in PersonId id)
raises(
      InvalidId);
```

A partir de un identificador de persona (ID) el servicio devuelve todos los atributos conocidos para dicha persona. *TraitNameSeq*, es una secuencia con los nombres de los atributos conocidos. Si el identificador no pertenece a ninguna persona, se genera la excepción *InvalidId*.

Método get_profile

```
Profile get_profile(
      in PersonId id,
      in SpecifiedTraits traits_requested)
raises(
      InvalidId,
            UnknownTraits,
            DuplicateTraits);
```

A partir de un identificador de persona (ID) el servicio devuelve todos los atributos solicitados en traits_requested (secuencia de nombres de atributos), mediante un Profile (secuencia de Traits).

Método get_profile_list

```
TaggedProfileSeq get_profile_list(
      in PersonIdSeq ids,
      in SpecifiedTraits traits_requested)
raises(
      TooMany,
      InvalidIds,
      DuplicateIds,
      UnknownTraits,
      DuplicateTraits);
```

Este método permite obtener perfiles para varios IDs al mismo tiempo, lo que resulta mucho más eficiente que el get_profile(), ya que con una sola operación se obtiene información de múltiples personas. Aparte de las ya mencionadas excepciones, *InvalidIds* se genera si uno o varios de los IDs introducidos no se corresponden con ninguna persona, y *DuplicateIds* si se introducen IDs repetidos.

Método get_deactivated_profile_list

```
TaggedProfileList get_deactivated_profile_list(
      in PersonIdSeq ids,
```

```
        in SpecifiedTraits traits_requested)
raises(
        NotImplemented,
        InvalidIds,
        DuplicateIds,
        UnknownTraits,
        DuplicateTraits);
```

Este método permite obtener perfiles de IDs cuyo estado sea deactivated (en desuso). Cuando un ID se encuentra en algún otro estado o no pertenece a ninguna persona, se lanza la excepción *InvalidId*. Si el servicio decide no conservar información de IDs en desuso se genera la excepción *NotImplemented*.

Método update_and_clear_traits

```
void update_and_clear_traits(
        in ProfileUpdateSeq profile_update_spec)
raises(
        InvalidIds,
        DuplicateIds,
        NotImplemented,
        MultipleTraits);
```

Este método se usa para modificar el perfil de un ID. La excepción *MultipleTraits* es similar a *DuplicateTraits* pero se aplica a operaciones en las que intervienen secuencias de IDs o de perfiles.

Método get_id_info

```
IdInfoSeq get_id_info(
        in PersonIdSeq ids)
raises(
        TooMany,
        DuplicateIds);
```

Devuelve el estado de cada ID introducido.

3 XML

3.1 Introducción

Una vez que hemos visto los tipos de datos que se van a intercambiar, resulta de gran importancia encontrar un mecanismo que nos ayude a serializar dichos datos y que posibilite su intercambio a través de la red. Para ello desde hace unos años se viene imponiendo el uso de XML (*eXtensible Markup Language*).

XML es un metalenguaje, es decir, un lenguaje para crear y describir otros lenguajes, lo que lo convierte en una herramienta extremadamente potente y flexible, ya que pueden crearse lenguajes y etiquetas específicos según las necesidades. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fácil y fiable.

3.2 Características de XML

Las características más importantes de XML son las siguientes:

- Permite mantener separados el contenido de la presentación:

La potencia y la belleza de XML residen en el hecho de que permite mantener la separación entre la interfaz de usuario y los datos estructurados. En XML se utilizan hojas de estilo, como el lenguaje de estilo extensible (XSL) y las hojas de estilo en cascada (CSS), para presentar los datos en un explorador.

- Fácilmente procesable:

XML es un formato basado en texto específicamente diseñado para almacenar y transmitir datos. Un documento XML se compone de elementos XML, cada uno de los cuales consta de una etiqueta de inicio (<title>), de una etiqueta de fin (</title>) y de los datos comprendidos entre ambas etiquetas (el contenido). Un

documento XML contiene texto anotado por etiquetas.

Los documentos XML se benefician de su estructura en forma de árbol o jerárquica, ya que estas estructuras son, en general, rápidas de acceder porque podemos trasladarnos a la parte que necesitemos, como pasando a través de una tabla de contenidos.

- Extensible:

Dado que la estructura de los documentos XML es igual en toda organización y para todos los usuarios de Internet, existirá la capacidad de buscar y manipular datos independientemente de las aplicaciones en las que se encuentren. Una vez encontrados los datos, se pueden entregar a través de la red y presentar en un explorador de muy distintas formas, o bien se pueden pasar a otras aplicaciones para procesarlos y verlos.

- Permite validar el contenido respecto a una gramática:

Con XML se pueden incluir archivos de definición de tipo de documento (DTD) con un documento para definir sus normas, así como los elementos que están presentes y la relación estructural entre los mismos. Los archivos DTD ayudan a validar los datos cuando la aplicación receptora no tiene una descripción incorporada de los datos entrantes. No obstante, los DTD son opcionales con XML.

Así, podemos distinguir dos tipos de documentos XML:

- **Documento bien/mal formado:** un documento se considera bien formado cuando sigue las reglas de sintaxis definidas por la especificación XML.

- **Documentos válidos/no válidos:** si un documento bien formado sigue las reglas definidas en un DTD o esquema, ese documento es válido.

Con el XML válido y bien formado, los datos codificados en XML son *autodescriptivos*, pues las etiquetas descriptivas están entremezcladas con los datos.

El formato abierto y flexible utilizado por XML permite su uso en cualquier lugar donde sea necesario intercambiar y transferir información. Esta capacidad lo hace extremadamente potente.

- Actualizaciones granulares:

Los datos se pueden actualizar de forma granular con XML, por lo que no es necesario volver a enviar un conjunto completo de datos estructurados cada vez que cambia parte de dichos datos. Sólo es preciso enviar el elemento modificado del servidor al cliente, y los datos modificados se pueden presentar sin necesidad de actualizar toda la interfaz de usuario.

- Es independiente de la plataforma sobre la que trabaja, con lo que simplifica el intercambio de datos.

3.3 Transformación Java-XML

Existen multitud de APIS de java que nos permiten trabajar con documentos XML, con diferentes funcionalidades. Nuestra principal finalidad es transformar documentos XML a objetos Java y viceversa. En el mundo Java la forma básica de abordar este problema es usar **JAXP** *(API Java para Proceso de XML)* para "parsear" el documento (usando SAX o DOM según convenga) y generar a mano los objetos, dando valor a sus atributos; en cualquier caso esta es una labor tediosa. En la actualidad contamos con una serie de herramientas de más alto nivel que nos permiten realizar esta tarea de una forma cómoda. Entre ellas podemos nombrar:

- **JAXB** (*Java Architecture for XML Binding*): realizada por SUN para la extracción de objetos Java a partir de entidades XML y viceversa.

- **XStream**: librería simple que permite pasar de documentos XML a objetos Java con una mínima configuración. Adicionalmente permite realizar el proceso contrario, es decir, serializar objetos Java en formato XML. Estas características lo hacen útil en escenarios en los que es necesario que la información sea persistente

sin recurrir a soluciones más complejas como sistemas de bases de datos.

- **Castor**: Es un framework (Java y de código abierto) para acceso a datos, que permite mapear objetos a documentos XML o bases de datos relacionales. Tiene una funcionalidad muy amplia.

- **XMLBeans**: Este proyecto de Apache todavía está en fase de desarrollo pero promete convertirse en una herramienta imprescindible para el tratamiento de XML. La idea es aprovecharse de la riqueza y las características de XML y del XML Schema traduciéndolas de la manera más natural posible a sus equivalentes Java y a las construcciones típicas del lenguaje.

- **Digester**: se inició como una parte de Jakarta Struts (un framework para el desarrollo de aplicaciones web J2EE). Su propósito inicial fue procesar el fichero de configuración de una aplicación web basada en Struts. Posteriormente, dada su evidente utilidad para una gran heterogeneidad de aplicaciones, fue incluido como un framework independiente dentro del proyecto Commons (un repositorio de componentes Java reutilizables). Nos oculta el proceso de "parseo" con JAXP y nos permite pasar directamente del XML a colecciones de objetos.

- **Skaringa**: API para enlazar Java y lenguaje XML. Transforma objetos Java a documentos XML y viceversa, y puede generar definiciones de schema XML para una clase Java. Aplicaciones típicas son intercambio de datos, persistencia de objetos, transformación de objetos...

Las transformaciones Java-XML que se desarrollan en este proyecto han sido realizadas mediante el uso de Castor, ya que así fueron implementadas por Mª José Caminero Marqués en su proyecto "Pasarela Java-XML para GPICS no clínicos y aplicación en un servidor demográfico" y es parte del código que se ha reutilizado para este proyecto. La elección de Castor para dichas transformaciones se debe a su gran funcionalidad y amplia implantación. Castor es básicamente el camino más corto entre: objetos java, documentos XML y tablas SQL. Además es importante destacar que Castor permite establecer los nombres de las etiquetas específicamente.

3.4 Castor XML

3.4.1 Introducción

Castor XML es un *framework* de enlace de datos a XML. A diferencia de las APIs XML DOM (*Document Object Model*) y SAX (*Simple API for XML*), que tratan con la estructura de un documento XML, Castor permite un trato con los datos definidos en un documento XML y un modelo de objetos que representan dichos datos. Castor también permite transformar casi cualquier objeto Java en un documento XML, y viceversa. En la mayoría de los casos el *framework* de transformación usa un conjunto de descriptores de clases y de campos (*ClassDescriptors* y *FieldDescriptors*) para describir cómo transformar un documento XML a objeto.

Se denomina ***marshal*** al acto de transformar un objeto a un *stream* (secuencia de bytes), en nuestro caso un objeto Java a un documento XML; el proceso contrario de convertir un *stream* a un objeto se conoce como ***unmarshal***.

3.4.2 El framework marshalling

El "framework marshalling", como su nombre implica, es responsable de hacer las conversiones entre Java y XML. El *framework* consta de dos clases, **org.exolab.castor.xml.Marshaller** y **org.exolab.castor.xml.Unmarshaller.**

Su uso es simple. Dado un objeto de tipo *bean* (ej. person), y un *FileWriter* (ej. writer, objeto de la clase *FileWriter* que nos permite escribir ficheros de caracteres) sobre el archivo XML que queremos obtener, realizamos la siguiente llamada:

Marshaller.marshal(person, writer); // Convierte de Java a documento XML.

Para el proceso contrario, necesitamos un *FileReader* (ej. reader, objeto de la clase *FileReader* que nos permite leer ficheros de caracteres) sobre el archivo XML que contiene los datos del objeto:

Person person=(Person)Unmarshaller.unmarshal(Person.class, reader);
//Convierte un documento XML en objeto.

Si queremos además usar un ***mapping*** (cuya funcionalidad se explicará más adelante) se debe crear y cargar antes de las llamadas anteriores:

```
//Se crea el manejador del mapa y se accede al archivo que lo contiene.
Mapping mapping = new Mapping();
mapping.loadMapping("mapping.xml");
Unmarshaller unmarshaller = new Unmarshaller(Person.class);
//Se carga el mapa en el transformador
unmarshaller.setMapping(mapping);
Person person = (Person)unmarshaller.unmarshal(reader);
```

3.4.3 Usando clases y objetos existentes

Castor puede transformar casi cualquier objeto a XML y viceversa. Cuando los descriptores no están disponibles usa la reflexión para obtener la información sobre el objeto. Si están disponibles, Castor los utiliza en la transformación.

Existen restricciones sobre los objetos que se pueden transformar. Es necesario que posean:

- Un constructor público por defecto (sin argumentos de entrada).
- Métodos "get" y "set" para todas las propiedades que se vayan a transformar.

3.5 Castor XML Mapping

3.5.1 Introducción

"Castor XML Mapping" es un modo de simplificar el enlace de clases Java a documentos XML. Permite transformar los datos contenidos en un modelo de objetos java a un documento XML, y viceversa.

También es posible confiar en el comportamiento por defecto de Castor para el proceso de transformación de los objetos a documentos XML, pero puede ser necesario ejercer más control sobre su funcionamiento. Por ejemplo, si el modelo de objetos Java ya existe, "Castor XML Mapping" puede usarse de puente entre el documento XML y dicho modelo, relacionando las clases y atributos Java con los elementos y atributos de un documento XML.

La información de un *mapping* está especificada en un documento XML. Este documento está escrito desde el punto de vista del modelo de objetos Java y describe cómo las propiedades de los objetos tienen que ser trasladadas a XML. Una restricción para el archivo *mapping* es que Castor no puede permitir la ambigüedad de cómo un atributo/elemento XML dado debe ser trasladado al modelo de objetos durante la transformación.

Es posible usar el *mapping* y el comportamiento por defecto de Castor en conjunción: cuando Castor tiene que encargarse de un objeto o de un dato XML pero no puede encontrar información sobre ellos en el *mapping*, se confiará en el comportamiento por defecto.

3.5.2 Transformación objeto Java - documento XML

Para Castor, una clase Java se mapea en un elemento XML. Cuando Castor transforma un objeto, será:

- Usando la información del mapa, el nombre del nuevo elemento XML será, si éste existe, el nombre del elemento XML, o
- Por defecto, usando el nombre de la clase como nombre del elemento XML.

Después se usa la información de los campos del archivo *mapping* para determinar cómo, para una propiedad dada de un objeto, se traslada a uno y sólo uno de los siguientes:

- Un atributo.
- Un elemento.
- Un texto de contenido.
- Nada, si elegimos ignorar ese campo en particular.

Este proceso es recursivo: si Castor encuentra una propiedad que tiene un tipo especificado de clase en otra parte del archivo *mapping*, se usará dicha información para transformar el objeto.

Por defecto, si Castor encuentra que no hay información para una clase dada en el *mapping*, se producirá la introspección en la clase y aplicará un conjunto de reglas para suponer los campos y hacerles la conversión. Las reglas por defecto son las siguientes:

- Todos los tipos primitivos, incluidos los tipos primitivos envueltos (*Boolean*, *Short*, etc.), serán transformados a atributos.
- Todos los otros objetos serán transformados a elementos con contenido de texto o elemento.

3.5.3 Transformación documento XML - objeto Java

Cuando Castor encuentra un elemento en un proceso de transformación de un documento, intentará usar la información del mapping para determinar qué objeto instanciar. Si no hay información en el *mapping* Castor usará el nombre del elemento para intentar adivinar el nombre de la clase e instanciarla (por ejemplo, para un elemento llamado 'test-element' Castor intentará instanciar una clase llamada 'TestElement'). Castor intentará entonces usar la información del campo del *mapping* para manejar el contenido del elemento.

Si la clase no está descrita en el *mapping* Castor intentará meterse en la clase usando la API de reflexión de Java para determinar si hay funciones de la forma getXxxYyy()/setXxxYyy (<type> x). Estos accesos están asociados con elementos/atributos XML llamados xxx-yyy. Castor sólo entrará en las variables de los objetos usando el método de acceso directo si no existen métodos get/set en la clase. En ese caso Castor buscará variables públicas de la forma:

public <type> xxxYyy;

y espera un elemento/atributo llamado 'xxx-yyy'. Las únicas colecciones que maneja como <type> son **java.lang.Vector** y **array** (versión superior a 0.8.10). Para <type> primitivos, Castor buscará primero un atributo y luego un elemento. Si <type> no es un tipo primitivo, Castor buscará primero un elemento y luego un atributo.

4 Arquitectura Cliente-Servidor

La arquitectura cliente-servidor consiste en una aplicación (el cliente) que solicita la realización de unas tareas a otra aplicación (el servidor) que puede estar o no en otra máquina. Para ello intercambian información en forma de mensajes. En el servidor se pueden ejecutar tareas comunes a varios clientes, facilitando la reusabilidad, el acceso a recursos compartidos, el mantenimiento y la gestión de servicios, el control de la seguridad, etc.

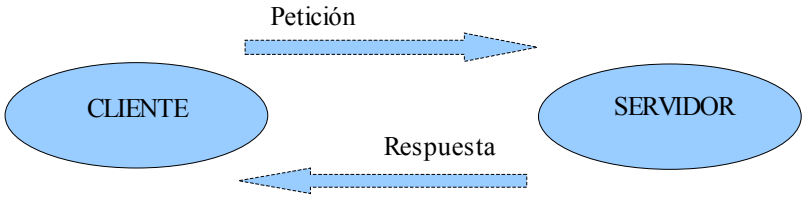

Figura 4.1: Interacción Cliente–Servidor

Los clientes realizan generalmente funciones como:

- ✔ Manejo de la interfaz de usuario.
- ✔ Captura y validación de los datos de entrada.
- ✔ Generación de consultas e informes sobre las bases de datos.

Por su parte los servidores realizan, entre otras, las siguientes funciones:

- ✔ Gestión de periféricos compartidos.
- ✔ Control de accesos concurrentes a bases de datos compartidas.
- ✔ Enlaces de comunicaciones con otras redes de área local o extensa.

Normalmente, pero no necesariamente, el cliente y el servidor están ubicados en distintas máquinas. Los clientes se suelen situar en ordenadores personales y/o estaciones de trabajo, y los servidores en procesadores departamentales o de grupo.

Entre las principales características de la arquitectura cliente-servidor se pueden destacar las siguientes:

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios internos en el servidor implican pocos o ningún cambio en el cliente.

5 Servicios Web

5.1 Definición

Un servicio web es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar información en redes de ordenadores como Internet. La interoperatividad se consigue mediante la adopción de estándares abiertos

Los servicios web pueden definirse, de forma genérica, como servicios ofrecidos vía web. Son aplicaciones que pueden ser publicadas, localizadas e invocadas a través de la web. En un primer paso, el proveedor del servicio publica un servicio web en un registro de servicios web (UDDI). Seguidamente un cliente que está buscando un servicio que se adapte a sus necesidades accede al registro obteniendo uno o múltiples resultados, de los que elige uno según sus preferencias. A continuación, el cliente descarga desde el servidor el documento WSDL que describe el servicio, con el que genera el código necesario para invocar el servicio.

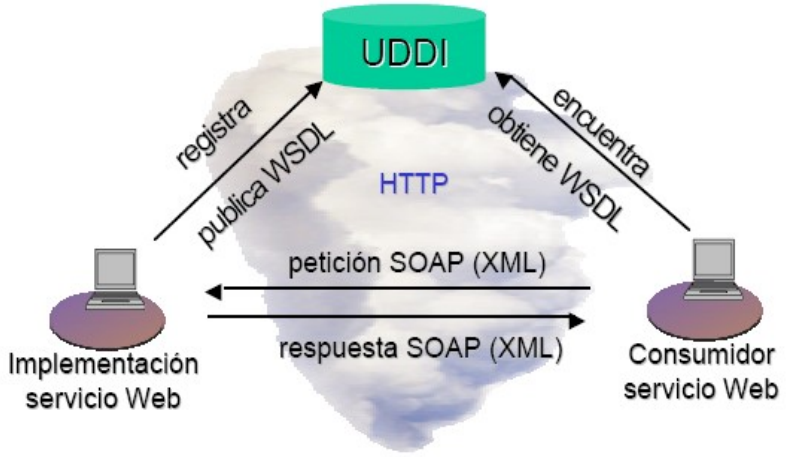

Figura 5.1: Definición esquemática de Servicio Web

Finalmente, tal y como se muestra en la figura 5.1, el cliente realizará una petición al servicio (dada una URL) usando el protocolo SOAP. El servicio recibe la petición, la procesa y devuelve una respuesta.

5.2 Características de los Servicios Web

Un Servicio Web es un componente software con las siguientes características:

- Es accesible a través del protocolo SOAP (*Simple Object Access Protocol*).
- Su interfaz se describe en un documento WSDL (*Web Service Desciption Language*).

SOAP es un protocolo ligero que define el intercambio de información en un entorno distribuido y descentralizado. Es un protocolo de mensajería XML (*eXtensible Markup Language*) utilizado para las comunicaciones en la arquitectura de servicios web. SOAP proporciona un mecanismo simple y consistente que permite a una aplicación enviar mensajes XML a otra aplicación. Un mensaje SOAP es una transmisión en un único sentido desde un emisor SOAP a un receptor SOAP, cualquier aplicación puede participar en este intercambio como emisor o receptor. SOAP es un protocolo de alto nivel que sólo define la estructura del mensaje y unas pocas reglas para su procesamiento. Es completamente independiente del protocolo de transporte subyacente. Actualmente el protocolo HTTP (*HiperText Transfer Protocol*) es el más utilizado para los mensajes SOAP.

WSDL es el lenguaje XML que se utiliza para describir la interfaz publicada por el servicio. Un documento WSDL proporciona toda la información necesaria para acceder y utilizar un servicio web: describe qué hace el servicio web, los tipos de datos que utiliza, cómo se comunica, y dónde reside.

UDDI (*Universal Description, Discovery and Integration*) es un catálogo independiente, basado en XML, que lista los negocios de Internet de todo aquel que se haya registrado. Publica la información de los servicios web y permite a las aplicaciones

comprobar qué servicios web están disponibles.

5.3 Ventajas e Inconvenientes de los Servicios Web

Algunas de las ventajas de los servicios web son:

- Aportan interoperatividad entre aplicaciones de software independientes en cuanto al lenguaje de programación o de las plataformas sobre las que se instalen.

- Los servicios web fomentan los estándares y protocolos basados en texto, lo cual facilita el acceso a su contenido y entender su funcionamiento.

- Al poder apoyarse en HTTP, los servicios web pueden aprovecharse de los sistemas de seguridad *firewall* sin necesidad de cambiar las reglas de filtrado.

- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

- Permiten la interoperatividad entre plataformas de distintos fabricantes por medio de protocolos estándar.

Y por último, se comentan algunas de los inconvenientes que presentan los servicios web:

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida, tales como RMI (*Remote Method Invocation*) o CORBA (*Common Object Request Broker Architecture*).

- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI o CORBA. Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.

- A poder apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en *firewall* cuyas reglas tratan de bloquear la comunicación entre programas a ambos lados del *firewall*.

- Existe poca información de servicios web para algunos lenguajes de programación.

5.4 Axis2

Axis2 es el proyecto de Apache que implementa el protocolo SOAP y que nos facilita el trabajo con servicios web. Es una implementación de código abierto de SOAP que proporciona un entorno de ejecución para servicios web implementados en Java.

Los servicios web, como ya se ha comentado, son sistemas software diseñados para soportar interoperatividad entre dos máquinas sobre una red. Para conseguir esta interoperatividad (independencia del lenguaje y del sistema operativo) los servicios web están basados en tecnologías estándares: XML, WSDL, SOAP y UDDI.

Axis2 nos facilita el desarrollo de aplicaciones basadas en servicios web proporcionando:

- La implementación del protocolo SOAP.

- Asistentes para trabajar con archivos WSDL.

- **Deserialización** del contenido XML (SOAP) en objetos Java que serán utilizados por el servicio y **serialización** de objetos Java que nos devuelva el servicio en XML (SOAP).

- Asistentes para la creación de esqueletos de servicio listos para ser desplegados de forma automática.

- Mejora de rendimiento respecto a Axis1 usando la tecnología **StAX** (*Streaming API for XML*), fusiona las ventajas de SAX (rendimiento) y DOM

(modelo “pull parsing”), sin sus inconvenientes (modelo “push parsing” de SAX y consumo excesivo de memoria de DOM).

- Soporte de modelo de programación síncrona y asíncrona.

- Soporta diferentes protocolos de transporte (HTTP, SMTP, TCP, etc.), siendo posible especificar en cada sentido de la comunicación un transporte diferente.

- El mapeo de 'XML/SOAP' a Java, y viceversa, se ha optimizado mediante **AXIOM** (*AXIs Object Model*) que usa StAX. Axiom es el API de bajo nivel de Axis2 para el manejo de mensajes SOAP.

- El mapeo de 'XML/SOAP' a 'Java-Beans', y viceversa, se puede implementar mediante varias tecnologías. En este proyecto, para tal labor se ha usado el *framework* Castor, como ya se ha comentado.

- Soporta la creación de servicios web mediante Spring.

- Soporte completo para SOAP con adjuntos (SwA, *SOAP with Attachments*)

En la figura 5.2 se muestra un esquema del ciclo de vida de un mensaje SOAP. La aplicación emisora crea el mensaje SOAP original, que consiste en un mensaje XML con cabeceras y un cuerpo. Si el sistema requiere el uso de recomendaciones relacionadas con servicios web (como por ejemplo WS-Security o WS-Addressing), el mensaje original puede necesitar un procesado adicional antes de abandonar el emisor. Una vez que el mensaje está listo es enviado a través de un protocolo de transporte como: HTTP, SMTP, etc. El mensaje se dirige al receptor, que lo recoge gracias al 'listener' (escuchador) del protocolo de transporte. (Si la aplicación no tiene el 'listener' activado, no recibirá ningún mensaje). De nuevo, si el mensaje requiere el uso de WS-Security u otras recomendaciones, se efectuará un procesado adicional. Finalmente, un despachador determina la aplicación específica a la que va dirigida el mensaje. A ella se le entregarán todos los datos transmitidos.

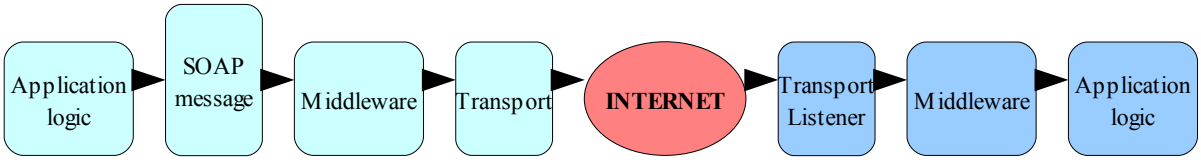

Figura 5.2: Ciclo de vida de un mensaje SOAP

En las figuras 5.3 y 5.4 se muestra cómo Axis2 gestiona los mensajes:

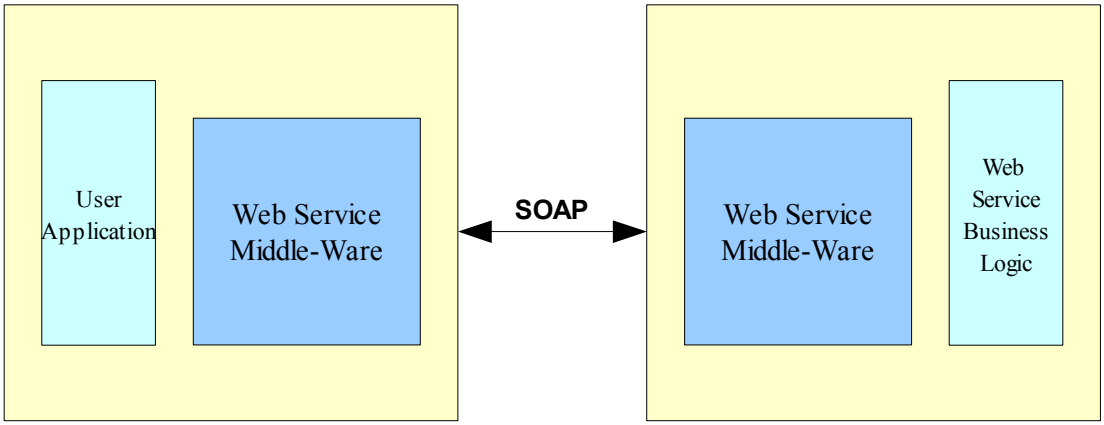

Figura 5.3: Gestión de los mensajes SOAP (I)

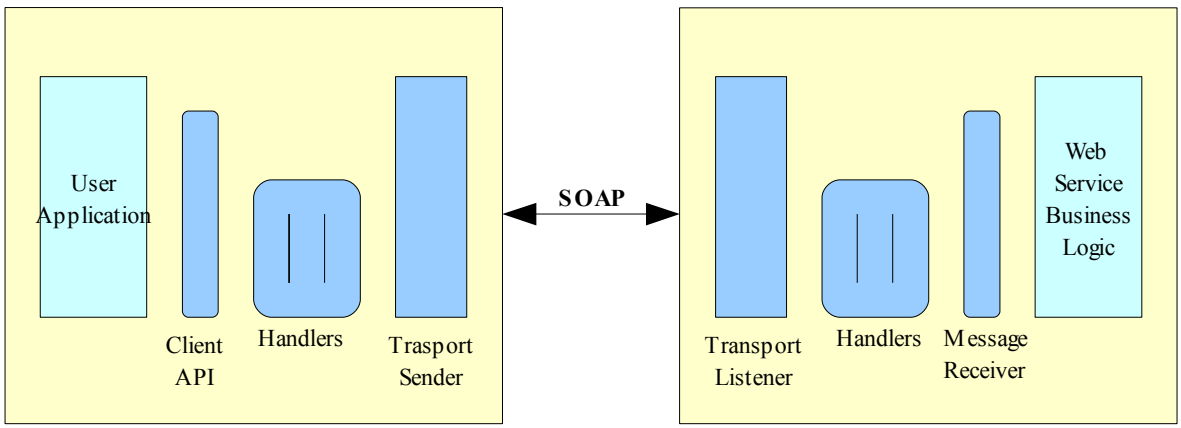

Figura 5.4: Gestión de los mensajes SOAP (II)

PARTE III : IMPLEMENTACIÓN PRÁCTICA

6 Herramientas utilizadas

6.1 Introducción

En este apartado de la memoria se comentan las herramientas usadas para la realización del proyecto. Se ha procurado utilizar en todo momento herramientas de software libre.

6.2 Lenguaje Java y JDK

Uno de los requisitos para el servicio web desarrollado era que fuera realizado en Java, ya que ofrece toda la funcionalidad de un lenguaje potente y podemos encontrar software reutilizable que nos facilita la tarea. Además, este proyecto es básicamente una ampliación y mejora de proyectos fin de carrera anteriores desarrollados en Java.

Lo primero que se necesita para desarrollar en Java es instalar el JDK (*Java Development Kit*). El JDK es un compilador y conjunto de herramientas de desarrollo para la creación de programas independientes y *applets* java. La versión que se instaló para el desarrollo de este proyecto es *"JDK1.6.0"*. Una vez instalado es muy importante crear a variable de entorno *JAVA_HOME* para indicar el directorio donde se ha instalado el JDK y añadir a la variable *PATH* dónde se encuentran los binarios del sistema (en el directorio *$JAVA_HOME/bin*) para poder ejecutarlos desde cualquier sitio.

Como entorno de desarrollo java se ha usado la aplicación Eclipse ("*Eclipse 3.2.1*") en su distribución para sistema operativo Linux.

6.3 Tomcat

Tomcat, también llamado *Jakarta Tomcat* o *Apache Tomcat*, funciona como un contenedor de aplicaciones desarrollado bajo el *Proyecto Yakarta* de la *Apache Software Fundation*, y es, por tanto, de código abierto. Además, implementa las especificaciones de

los servlets y de *JavaServer Pages* (JSP) de *Sun Microsystems*. La versión 5.5.20 de Tomcat, que es la usada en este proyecto, soporta las especificaciones "*Servlet 2.4*" y "*JavaServer Pages 2.0*".

Tomcat es un servidor web con soporte de servlets y JSPs. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache. Sin embargo, Tomcat puede funcionar como servidor web por sí mismo, y es así como se utiliza en este proyecto. En sus principios existió la percepción de que el uso de Tomcat de forma autónoma era recomendable sólo para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos de alto nivel de tráfico y alta disponibilidad.

Una vez que se ha configurado debidamente Tomcat, se ha creado la variable de entorno *CATALINA_HOME* con el directorio de instalación de Tomcat, y hemos arrancado el contenedor de aplicaciones, podemos acceder a la página de inicio de Tomcat a través de la URL: http://localhost:8080/ (el puerto configurado por defecto es el 8080).

6.4 Axis2

Axis2, como ya se comentó en la introducción teórica, es el proyecto de la *Apache Software Fundation* que implementa SOAP y nos facilita el trabajo con servicios web. Es una implementación de código abierto de SOAP que proporciona un entorno de ejecución para servicios web implementados en Java. Para el desarrollo de este proyecto se ha utilizado *"Axis2 1.2"*.

Para la instalación de Axis2 se utilizó la distribución WAR (*Web Archive*), una versión lista para ser desplegada como una aplicación web en un contenedor de servlets. En nuestro caso el contenedor de servlets (o de aplicaciones) es Tomcat. Para desplegar Axis2 sobre Tomcat basta con copiar el archivo WAR en el directorio *$CATALINA_HOME/webapps* de Tomcat. Una vez que tenemos Axis2 desplegado y configurado (con las variables de entorno *AXIS2_HOME* y *AXIS2_CLASSPATH*

debidamente inicializadas), ya podemos acceder a la página de inicio de Axis2 a través de la siguiente URL: http://localhost:8080/axis2 .

6.5 Axiom

Axiom (*AXIs Object Model)* es el API de bajo nivel de Axis2 para el manejo de mensajes SOAP, como ya se ha comentado en la introducción teórica, y está basado en el uso de StAX (*Streaming API for XML*). Fue diseñado con el objetivo de impulsar Axis2, ya que carga mínimamente el sistema, tanto en CPU como en memoria.

En este proyecto se ha usado la versión 1.2.4 de Axiom. En el apartado 9 de la memoria se entrará más en detalle en la gran utilidad de Axiom para el desarrollo de servicios web y el intercambio de mensajes SOAP.

6.6 Otras Herramientas

En los apartados anteriores ya se han comentado las herramientas y tecnologías más importantes que se han usado para el desarrollo de este proyecto. Además de las ya mencionadas, podemos comentar que se ha usado una base de datos MySQL ("*MySQL 5.0.4* "), así como phpMyAdmin para agilizar el manejo y la gestión de la base de datos.

También se ha utilizado "*Visual Paradign for UML 5.0*" para el modelado de la estructura de la aplicación en diagramas UML, la herramienta de gestión de proyectos Planner para la creación del diagrama de Gantt con las distintas fases de la implementación, y por último JavaDoc, una herramienta para generar documentación de código en formato HTML a partir de comentarios en el código fuente.

7 Situación de Partida

7.1 Introducción

Para la implementación de este proyecto se ha partido del proyecto fin de carrera "Pasarela Java-XML para GPICS no clínicos y aplicación en un servidor demográfico" realizado por Mª José Caminero Marqués, el cual, a su vez, fue implementado partiendo del proyecto "Servidor demográfico basado en normas del CEN y CORBAmed" realizado por Mª Ángeles Repullo López.

El propósito de este último era el desarrollo de un sistema de información demográfica y de un servidor de acceso a la misma, que cumpliera con los estándares europeos de información sanitaria indicados en la introducción teórica. Partiendo de este sistema de información inicial, Mª José amplió los tipos de datos soportados y mejoró la interfaz de comunicación mediante la conversión a documentos XML como un paso más hacia su integración en un servicio web.

A continuación, se va a detallar la situación inicial del proyecto. Comentar, que se va a mantener prácticamente la misma estructura de paquetes existentes, y las clases de nueva implementación se han ido añadiendo en otros paquetes aparte de los ya existentes.

7.2 Librería de Datos

La librería de datos contiene la implementación de los tipos del CEN (es decir, los tipos básicos y los tipos GPICS), los cuales se detallan en los apartados 3.3 y 3.4 de la memoria del proyecto de partida ("Pasarela Java-XML para GPICS no clínicos y aplicación en un servidor demográfico" realizado por Mª José Caminero Marqués).

Cada tipo se encuentra implementado como una clase Java, y más concretamente con características de *bean*, es decir:

- Todos los atributos son privados.

- Existen métodos de acceso "get" y "set" públicos para todos los atributos.

- Un constructor vacío y, como mínimo, otro constructor con todos los atributos de la clase.

- Método "toString()" para la representación por pantalla.

Los tipos básicos del CEN se han tomado del paquete **BasicTypes** del proyecto de partida. Todas las clases son públicas, y heredan de la clase abstracta **DV** (*DataValue*). Las clases que representan a los tipos básicos compuestos tienen como variables tipos primitivos o incluso otros tipos compuestos. Las clases contienen además el siguiente método:

```
public boolean fromXML2Object(Mapping mapa, String documentoXML)
```

que permite crear un objeto Java a partir del documento XML, sin conocer la clase concreta. Esto se debe a que es un método sobrescrito por todas las clases del paquete. Así, si se crea un objeto *DV,* del cual heredan todas las clases, se puede invocar éste método, y se creará el objeto de la clase contenida en el documento XML con todos sus campos rellenos con los datos del XML. Basa su funcionamiento en el framework de Castor y se le pasan como parámetros un objeto **org.exolab.castor.mapping.Mapping**, que especificará a Castor como traducir el documento a objeto; y un *String* con el URI del documento XML que se quiere transformar a objeto.

Los tipos GPICS del CEN se han tomado del paquete **GPICS** del proyecto inicial, dentro del cual se han diferenciado por su significado en 2 subpaquetes: GPICS.CommonSubComponents y GPICS.CAG_GPICS. Cada tipo se encuentra encapsulado en una clase Java con características de *bean*, teniendo como variables tipos simples o incluso tipos complejos CAG.

7.3 Base de Datos

La base de datos se encontraba totalmente implementada, así como su generación a partir del código y su manejo mediante clases Java. Todas las clases desarrolladas para interactuar con la base de datos se encontraban en el paquete **Local.PID.DataBase**. Sin embargo, se tuvieron que realizar algunos cambios en estas clases para poder adaptar la base de datos al entorno del este proyecto.

El *driver* que se utilizaba era el Driver JDBC-ODBC (sun.jdbc.odbc.JdbcOdbcDriver). JDBC (*Java DataBase Conectivity*) es un estándar para manejar bases de datos en Java, y ODBC (*Object DataBase Conectivity*) es un estándar de Windows para manejar bases de datos de forma que cualquier programa en Windows que desee acceder a bases de datos genéricas debe usar este estándar.

Para la realización de este proyecto, como ya se ha comentado en apartados anteriores, se optó por utilizar una base de datos MySQL, por ser una de las más usadas en entornos UNIX y por su fácil y sencillo manejo. Para adaptar el código existente a nuestro entorno con el sistema operativo Linux se tuvieron que realizar los siguientes cambios en el código inicial:

- En la clase **DB_Manager.java**:

Atributos	Proyecto de partida	Proyecto actual
url	"jdbc:odbc:PersonDB"	"jdbc:mysql://localhost/PersonDB"
driver	"sun.jdbc.odbc.JdbcOdbcDriver"	"com.mysql.jdbc.Driver"
user	""	"userpids"
password	""	"passwordpids"

Tabla 7.1: Cambios realizados en la clase DB_Manager.java

"PersonDB" es el nombre que se le ha dado a la base de datos creada. Para acceder a la base de datos se ha creado un usuario, con nombre "userpids" y clave "passwordpids". Anteriormente se conectaba a la base de datos sin ningún usuario ni clave, pero parecía más lógico y seguro restringir el acceso a la misma. Por ello, se modificó el código para tal efecto.

- En la clase **DB_Factory.java**:

Proyecto de partida	Proyecto actual
BIT	VARCHAR(1)
TIMESTAMP	DATETIME
use	`use`
COUNTER	VARCHAR(32)

Tabla 7.2: Cambios realizados en la clase DB_Factory.java

La tabla 7.2 muestra los cambios realizados en el tipo de algunos atributos de las tablas creadas en la base de datos. La palabra ***use*** es una palabra reservada en Mysql y se usaba como nombre de un atributo en la tabla *TelecomUse*, por lo que se tuvo que entrecomillar para que no diera problemas.

Además en la clase **DB_Person.java** se hacían consultas a algunas tablas con el primer carácter del nombre de la tabla en minúscula (en lugar de mayúscula) y como MySQL es sensible a mayúsculas y minúsculas, se tuvieron que revisar los nombres de las tablas para que en todas las consultas estuvieran con el primer carácter en mayúscula.

7.4 Pasarela Java-XML mediante el uso de Castor

Como ya se ha comentado en apartados anteriores de la memoria, en el proyecto de partida se había implementado una pasarela Java-XML mediante el uso de Castor. Se pensó en reutilizar este código para la realización del servicio web.

La conversión Java-XML se realiza mediante el uso de 3 mapas:

- **basicTypes.xml**: para la conversión de los tipos básicos en el paquete **BasicTypes**.

- **GPICS.xml** : para la conversión de los tipos GPICS implementados en el paquete **GPICS**.

- **PIDs.xml**: para los tipos de las estructuras básicas del PID implementadas en el paquete **PID.DataTypes**.

Para la transformación de Java a XML se utilizan siempre las mismas líneas de código:

```
//Creamos el mapping y lo cargamos en su controlador.
Mapping mapa= new Mapping();
mapa.loadMapping("Directorio raíz del proyecto/public_html/PIDs.xml");
File documentoXML=new File(URI);
FileWriter file_writer_documentoXML= new FileWriter(documentoXML);
//Creamos el elemento encargado de transformar de objeto a XML
Marshaller m= new Marshaller(file_writer_documentoXML);
m.setMapping(mapa);
m.marshal(objeto_java);
```

El mapa a cargar será el que contenga al objeto java a transformar.

Para la transformación de XML a Java es importante resaltar que cada documento XML contendrá un único objeto java de los implementados, lo cual se corresponde con el principio de XML de que cada documento tenga un único elemento raíz. Además, para la obtención del objeto se conocerá a priori la clase del objeto contenido en el documento XML. Dicha clase la conocemos en este proyecto porque las interfaces para el intercambio

de información están completamente definidas, de modo que se sabe en cada momento el objeto contenido en el documento XML a tratar. Así dependiendo del procedimiento en el que estemos se determinará el objeto raíz contenido en el XML, y se podrá usar el siguiente código:

```
//Creamos el mapping y le cargamos el archivo del mapa
Mapping mapa=new Mapping();
mapa.loadMapping("Directorio raíz del proyecto/public_html/PIDs.xml");
//Documento de entrada que contiene al objeto
File documentoXML= new File(URI);
//Lector del documento XML
FileReader in = new FileReader(documentoXML);
//Se encarga de la transformacion xml a objeto
Unmarshaller un= new Unmarshaller(Clase_del_Objeto.class);
un.setMapping( mapa );
//Transformación a objeto
Clase_del_Objeto nuevoObjeto = (Clase_del_Objeto) un.unmarshal(in);
```

donde Clase_del_Objeto debe ser reemplazado por la clase del objeto correspondiente.

7.5 PIDS de CORBAmed

Una vez que disponemos de la librería de datos que necesitamos y estudiado el modo de realizar la pasarela Java-XML, se continuó con el código del PIDS de modo que se transformen las estructuras de datos (recuperadas de la base de datos del proyecto de partida) a documentos XML y viceversa.

Dentro de la documentación del PIDS, como se ha comentado en la introducción teórica, se definen una serie de estructuras de datos que se usan en sus interfaces. Dichas estructuras fueron implementadas en el proyecto de partida dotándolas de estructura de *bean* y fueron incluidas en el paquete **PID.DataTypes**.

En el paquete **PID.Exception** fueron implementadas las distintas excepciones que se pueden producir al invocar a los métodos de las interfaces del PIDS: *DuplicateIds*, *DuplicateTraits*, *InvalidId*, *InvalidIds*, *InvalidWeight* y *UnknownTraits*.

A su vez, existía la implementación de IdentifyPerson como **IdentifyPersonImpl**, y ProfileAccess como **ProfileAccessImpl**, las cuales se encuentran en el paquete **Local.PID.ServiceClasses** junto con las interfaces que las definen (**IdentifyPerson** y **ProfileAccess**).

Y por último, en el paquete **PID.Implementation**, se encontraban las tres clases principales que realizan de manera automática las conversiones Java a XML y viceversa, necesarias para la ejecución del código: **Canonizador**, **IdentifyPersonXML** y **ProfileAccessXML**.

La clase *Canonizador* se crea con el fin de codificar los documentos XML para que a priori no contengan caracteres no válidos como pudieran ser ñ o vocales con tilde. Los datos recuperados de la base de datos están correctamente escritos en castellano, pero debido a que ni tildes ni ñ se consideran caracteres válidos, se ha optado por una codificación de los mismos. Esta canonización consiste en que cada vez que aparece uno de los caracteres no válidos se sustituirá por # seguido del carácter más equivalente, es decir: á por #a, é por #e, í por #i, ó por #o, ú por #u, ñ por #n, Á por #A, É por #E, Í por #I, Ó por #O, Ú por #U, Ñ por #N; además, para el caso de que en documento original existiera un símbolo #, este será duplicado, de modo que # se sustituye por ##. Este proceso se debe realizar al contrario para aquellos documentos que son recibidos por los métodos.

La clase *Canonizador* fue revisada y modificada ligeramente debido al cambio en la codificación del proyecto de partida a ISO-8859-1. Cada carácter con acento o 'ñ' tuvo que ser reescrito.

La clases *IdentifyPersonXML* y *ProfileAccessXML* se encargan de envolver a *IdentifyPersonImpl* y *ProfileAccessImpl*, respectivamente, de modo que las entradas y las salidas sean documentos XML. Estas dos clases se han revisado para este proyecto, cambiándose las rutas de los documentos XML que se crean durante la ejecución de la aplicación.

A continuación se muestra de forma esquemática cómo funcionan los métodos implementados en las clases *IdentifyPersonXML* y *ProfileAccessXML*:

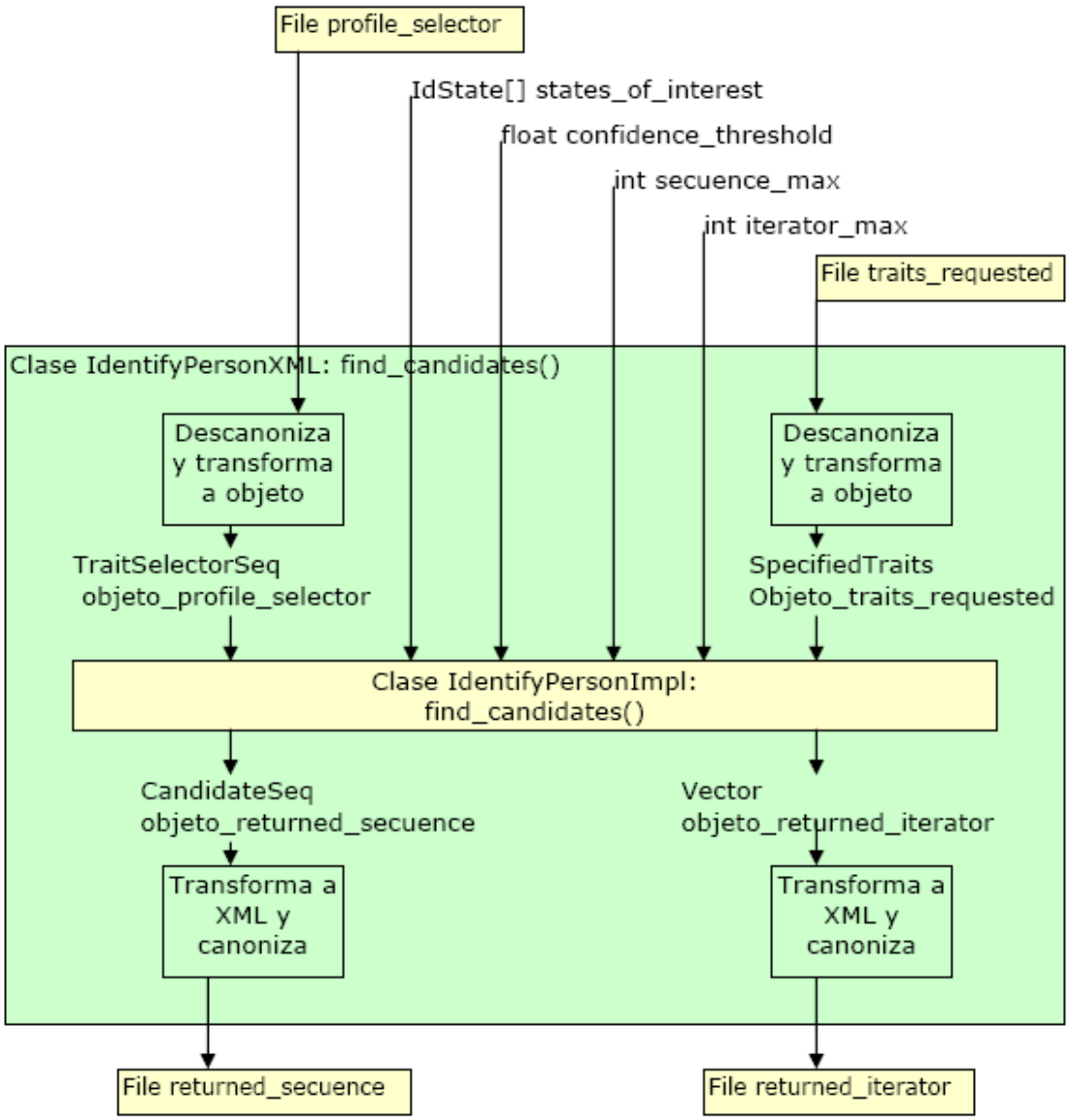

Figura 7.1: Diagrama del método find_candidates() de la clase IdentifyPersonXML

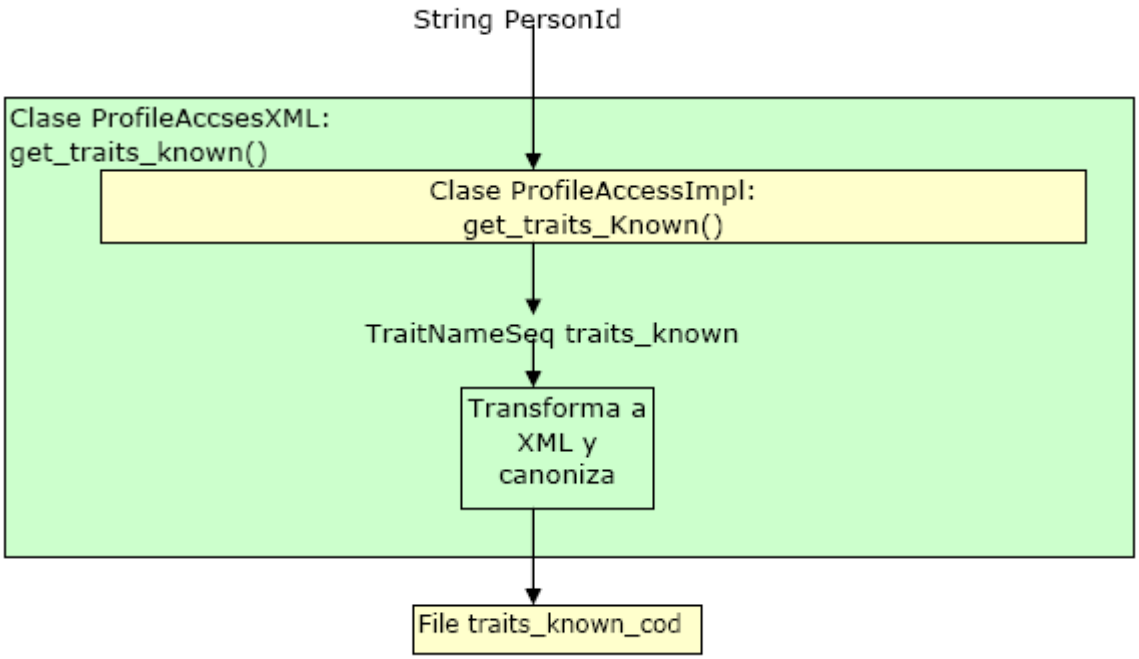

Figura 7.2: Diagrama del método get_traits_known() de la clase ProfileAccessXML

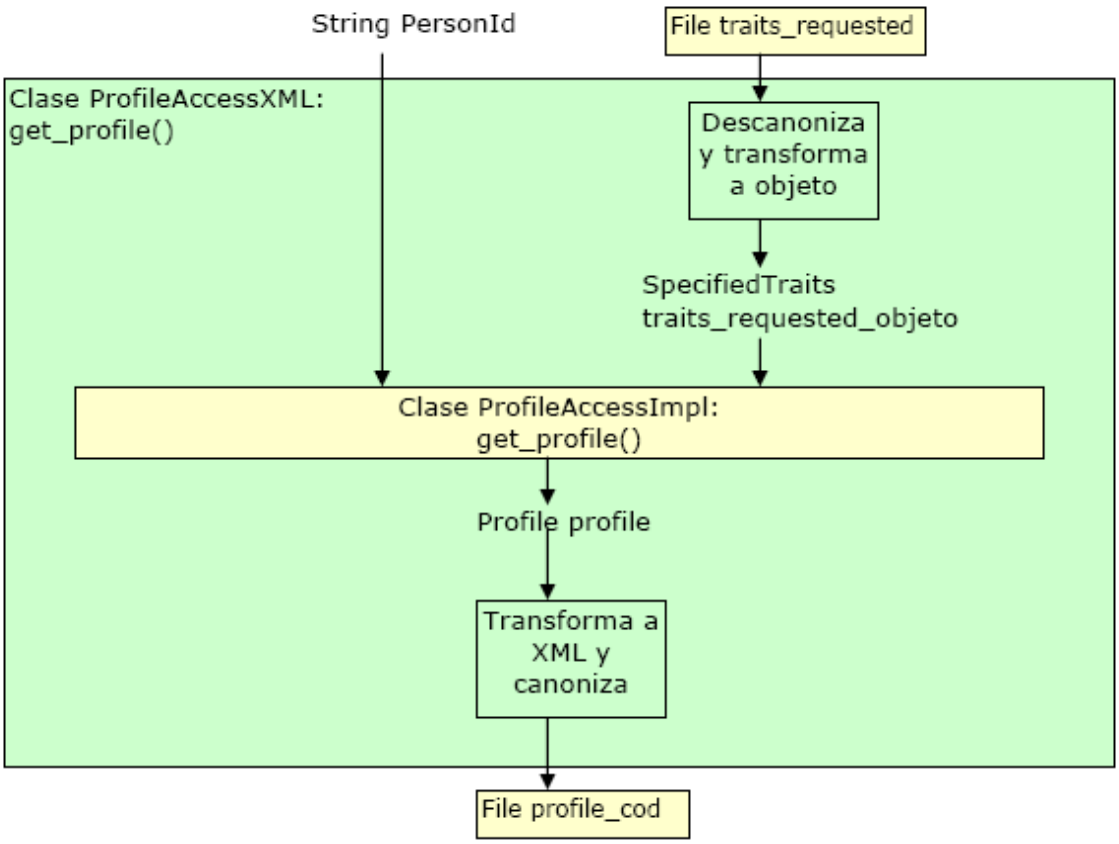

Figura 7.3: Diagrama del método get_profile() de la clase ProfileAccessXML

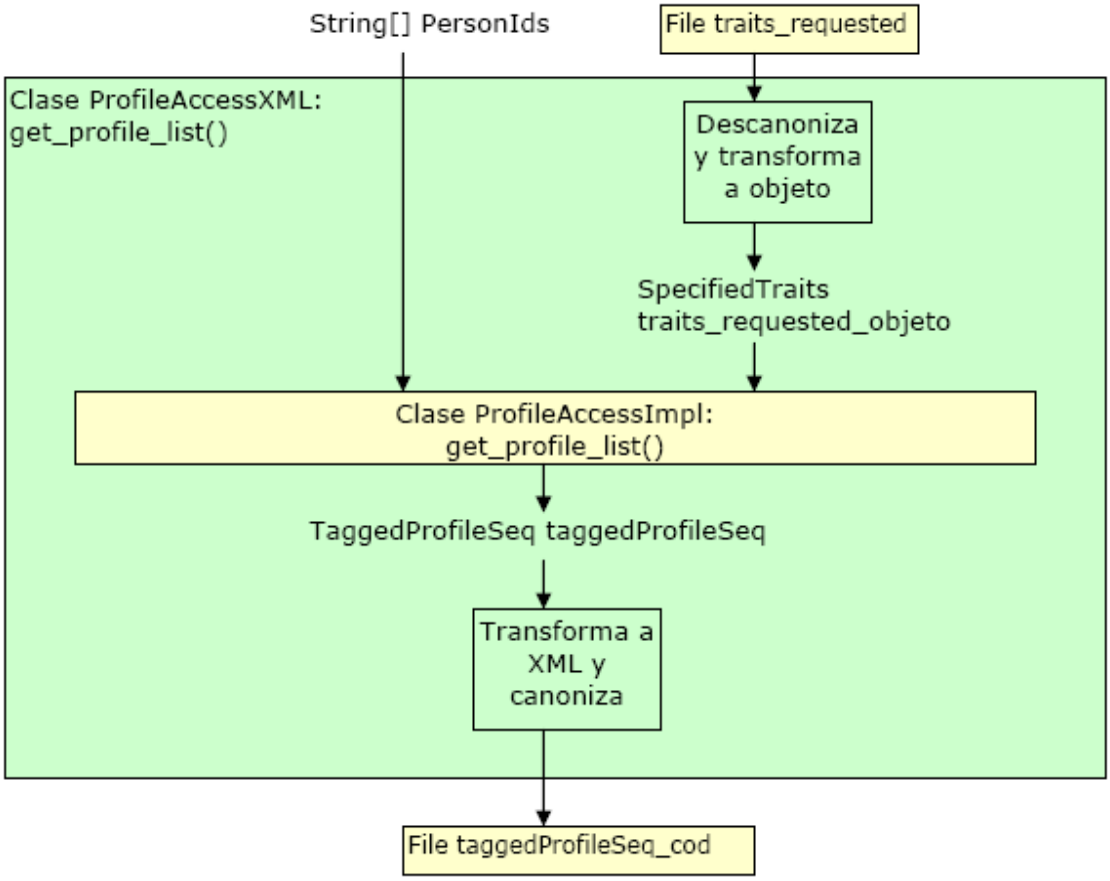

Figura 7.4: Diagrama del método get_profile_list() de la clase ProfileAccessXML

8 Implementación

8.1 Servidor

La realización de un Servicio Web requiere trabajar con el protocolo SOAP, lo que implica muchas tareas engorrosas y propensas a errores como, por ejemplo, tener que construir íntegramente los mensajes transmitidos. Afortunadamente existen herramientas como Axis2, que permiten trabajar a más alto nivel sin tener que entrar en los entresijos de SOAP. Además, Axis2 encaja perfectamente con la idea de flexibilidad e independencia porque está escrito en Java y , al formar parte del Proyecto Apache, es software de código abierto, pudiendo obtenerse de forma gratuita.

8.1.1 Estructura y Funcionamiento del servidor

El funcionamiento del servidor en líneas generales es el siguiente:

1. Llega una petición (HTTP) que es recibida por el contenedor de aplicaciones (Tomcat).

2. El contenedor de aplicaciones decide a qué aplicación de las que tiene desplegadas corresponde la petición (en este caso Axis2) y se la pasa.

3. Axis2 extrae la petición SOAP de la petición HTTP y decide a qué servicio de los que tiene desplegados corresponde el método al que va dirigida la petición, y por último, lo invoca pasándole como parámetros los que se indican en la petición.

4. El método invocado se ejecuta, siendo Axis2 quien recibe sus resultados.

5. Axis2 construye el mensaje SOAP de respuesta y se lo pasa al contenedor de aplicaciones.

6. El contenedor de aplicaciones construye la respuesta HTTP y se la envía al cliente que solicitó la petición.

Cuando hablamos de petición, aclarar que puede ser para cualquiera de los cuatro métodos implementados del PIDS. De la interfaz **IdentifyPerson** se ha implementado el método *find_candidates*, y de la interfaz **ProfileAccess** se han implementado los métodos *get_traits_known*, *get_profile* y *get_profile_list*.

En la figura 8.1 se muestra la estructura de directorios del servidor. En **/documentos** se almacenan los documentos XML de entrada y de salida del código (así como los ficheros XML intermedios, es decir, sin codificar), en **/lib** las librerías que utiliza el servidor y en **/public_html**, al igual que en el proyecto de partida, se almacenan los mapas necesarios para las transformaciones Java-XML mediante el uso de Castor.

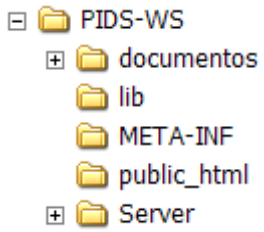

Figura 8.1: Estructura de directorios del servidor

En la carpeta **/META-INF** se encuentra el archivo descriptor del servicio *services.xml,* que contiene la configuración del servicio web, y un archivo WSDL por cada una de las interfaces del PIDS que se implementan (*IdentifyPerson.wsdl* y *ProfileAccess.wsdl*). Todo servicio web desplegado en Axis2 debe tener su configuración en un archivo *services.xml,* cuya estructura se mostrará más adelante.

A continuación, tal y como se muestra en la figura 8.2, se detalla la estructura de paquetes java del directorio **/Server**, donde se encuentran todas las clases java que se implementan en el servidor.

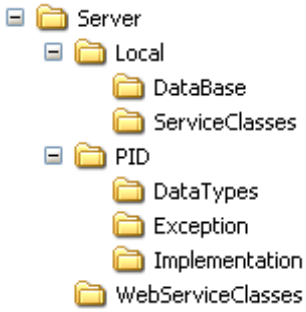

Figura 8.2: Estructura de paquetes del servidor

- **Server.Local.PID.DataBase**: Este paquete contiene las clases necesarias para que el servicio pueda interactuar con la base de datos y realizar búsquedas de atributos e identificadores de persona.

- **Server.Local.PID.ServiceClasses**: En este paquete encontramos las clases necesarias para que el servicio pueda acceder a los métodos que implementan las interfaces *IdentifyPerson* y *ProfileAccess* del Sistema de Identificación de Personas (PIDS) de CORBAmed (*find_candidates*, *get_traits_known*, *get_profile* y *get_profile_list*), a partir de las cuales se realizan las consultas a la base de datos para obtener identificadores de persona o valores de atributos.

- **Server.PID.DataTypes**: Aquí se encuentran las clases que encapsulan a los tipos de datos definidos en el PIDS en el apartado de *Data Types*.

- **Server.PID.Exception**: Este paquete contiene las clases que encapsulan a las excepciones definidas en el PIDS en el apartado de *Data Types*.

- **Server.PID.Implementation:** Este paquete contiene las clases necesarias para que el servicio pueda realizar sus funciones de pasarela Java-XML.

- **Server.WebServiceClasses**: En este paquete están todas las clases que ofrecen las interfaces del PIDS a través del Servicio Web.

Antes de continuar aclarar que el servidor recibe los documentos XML de entrada canonizados, es decir, sin acentos ni caracteres especiales como la 'ñ'. Estos documentos de entrada se crean en **/documentos/entradas**. Posteriormente, dado que la base de datos de ejemplo sí contiene acentos y demás, estos documentos XML se descanonizan y se almacenan, junto con los documentos de salida, en **/documentos/intermedios**. Y por último, antes de mandar el contenido de los documentos de salida al cliente, se canonizan, almacenándose en **/documentos/salidas**.

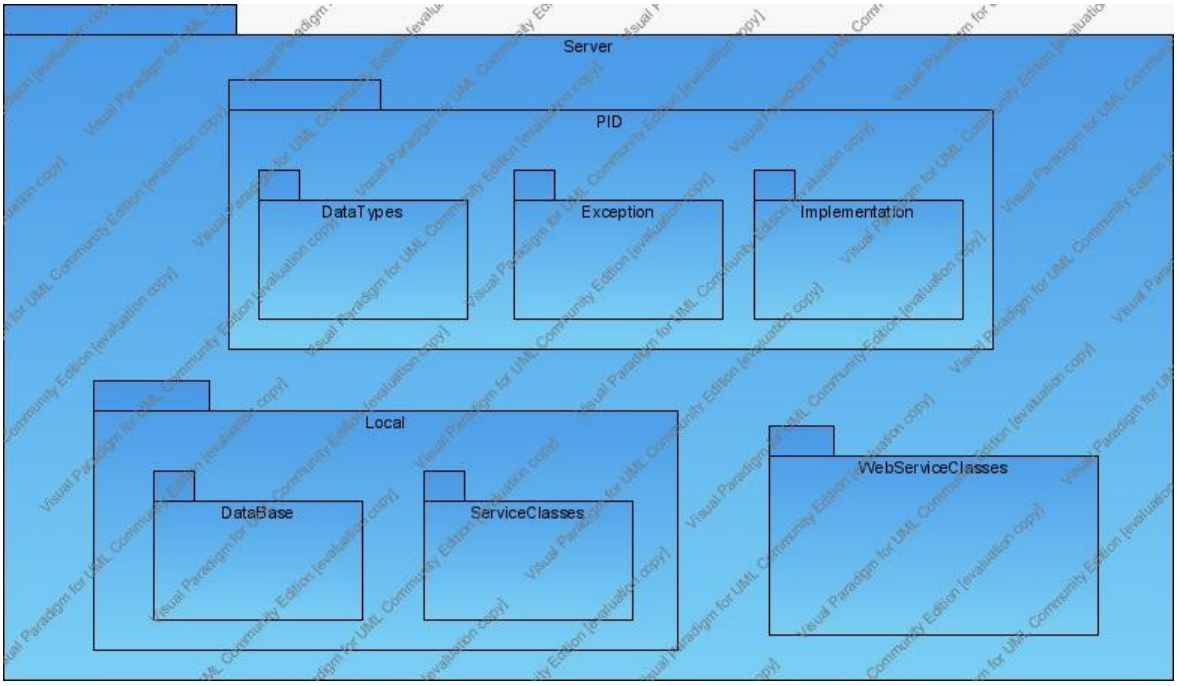

Diagrama 8.1: Diagrama de Paquetes del Servidor

Como se ha comentado en el apartado de la memoria "Situación de Partida", se ha reutilizado parte del código del proyecto fin de carrera "Pasarela Java-XML para GPICS no clínicos y aplicación en un servidor demográfico" para el desarrollo del servicio web implementado, manteniendo prácticamente la misma estructura del proyecto original más el paquete *Server.WebServiceClasses.*

Dado que el paquete *Server.WebServiceClasses* ha sido el paquete más relevante para el desarrollo del Servicio Web es en el que nos detendremos a explicar con más detalle.

8.1.2 Creación de un Servicio Web

Antes de explicar en detalle las clases creadas para la implementación del servicio web, se van a enumerar muy brevemente qué pasos hay que seguir para la creación de un servicio web y su posterior despliegue en Axis2. De esta forma, resultará más fácil comprender todo lo demás.

Los pasos a seguir son los siguientes:

1. Creación de la clase que define la interfaz del servicio, con cada método que tendrá como argumento un objeto de la clase **org.apache.axiom.om.OMElement**. Un *OMElement* es la forma que tiene Axiom (*Axis2 Object Model*) de representar un elemento XML.

2. Creación de la clase java que implementa la interfaz del servicio.

3. Creación del archivo descriptor del servicio, *services.xml,* es decir, el fichero de configuración del servicio web.

4. Creación del fichero .wsdl que describe la interfaz del servicio.

5. Creación del fichero .aar, con las clases java en el paquete adecuado, y el fichero *services.xml* y el .wsdl ambos en el directorio */META-INF*. El fichero .aar debe crearse desde el directorio raíz del servidor.

6. Desplegamos el servicio web en Axis2 pegando el fichero .aar en el directorio *$AXIS2_HOME/WEB-INF/services.*

En nuestro caso, como el servicio va a estar formado por dos interfaces, *IdentifyPerson* y *ProfileAccess,* tendremos dos ficheros .wsdl en el directorio /META-INF, *IdentifyPerson.wsdl* y *ProfileAccess.wsdl*. Y el fichero *services.xml* será más complejo de lo habitual, ya que será como estar implementando dos servicios.

8.1.3 Paquete Server.WebServiceClasses

8.1.3.1 Diagrama de Clases

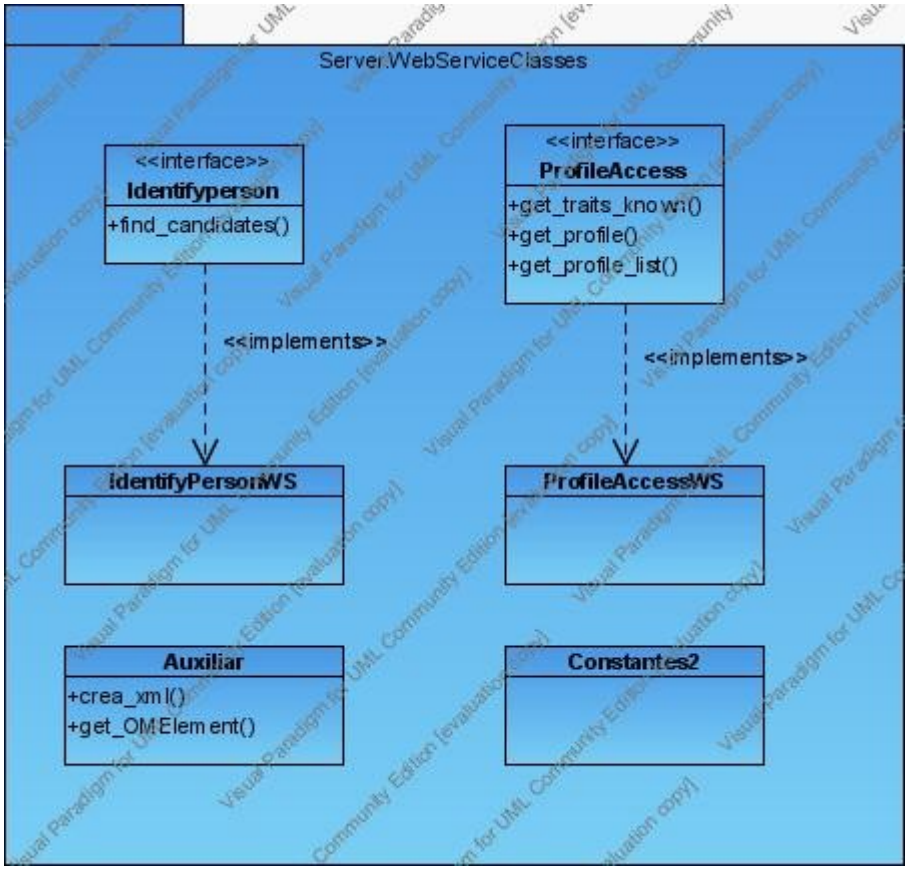

Diagrama 8.2: Diagrama de Clases del paquete Server.WebServiceClasses

8.1.3.2 IdentifyPerson.java

En esta clase se define la interfaz **IdentifyPerson** del Servicio Web. Es la clase encargada de ofrecer la interfaz *IdentifyPerson* del Servicio de Identificación de Personas (PIDS) a través del servicio web. Como sabemos, una interfaz contiene una colección de métodos que se implementan en otro lugar. En nuestro caso, la interfaz *IdentifyPerson* posee un sólo método llamado ***find_candidates***.

Como se puede apreciar, los nombres de las interfaces, así como los nombres de los métodos implementados para ofrecer el Servicio de Identificación de Personas (PIDS) a través del servicio web coinciden con los que aparecen en la especificación del PIDS. De esta forma, resulta más fácil intuir qué hace cada interfaz y cada método implementado en el servicio.

8.1.3.3 IdentifyPersonWS.java

Esta clase implementa la interfaz **IdentifyPerson** del servicio web. La clase **IdentifyPersonWS** se encarga de envolver a la clase **IdentifyPersonXML** implementada en el proyecto de partida. A su vez, la clase *IdentifyPersonXML* (del paquete *Server.PID.Implementation*) se encarga de envolver a **IdentifyPersonImpl** (del paquete *Server.Local.PID.ServiceClasses*) de modo que las entradas y las salidas sean ficheros XML.

El método implementado en esta clase es:

```
public OMElement find_candidates (OMElement peticion)
```

El cliente invocará este método cuando conociendo cierta información que identifique a una persona quiera que el servidor le devuelva los candidatos que más se correspondan a ese perfil. El método recibe un OMElement y devuelve otro OMElement como salida. A través del OMElement de entrada recibimos todos los datos que necesitamos para invocar al método *find_candidates* de la clase *IdentifyPersonXML*. Es decir, el contenido de dos documentos XML (***traits_requested_cod.xml*** y ***profile_selector_cod.xml***, los cuales se crean en la carpeta *documentos/entradas*), y los parámetros **confidence_threshold**, **sequence_max** e **iterator_max**, en el mismo orden se definen en el método *find_candidates* que se define en la especificación del PIDS. El fichero *profile_selector_cod.xml* contiene una serie de atributos con sus correspondientes valores para hacer la búsqueda de los candidatos y el fichero *traits_requested_cod.xml* contiene los nombres de los atributos solicitados para cada candidato.

El OMElement de salida estará formado por el contenido de los dos archivos

XML que se reciben de la ejecución del método *find_candidates* de la clase *IdentifyPersonXML*, ***returned_sequence_cod.xml*** y ***returned_iterator_cod.xml*** (que se crean en *documentos/salidas)*. El fichero *returned_sequence_cod.xml* contendrá los candidatos devueltos, tantos como indique **sequence_max**. Si se encuentra un número mayor de candidatos, se devuelven dentro del fichero *returned_iterator_cod.xml*, pudiéndose indicar el tamaño máximo de éste mediante **iterator_max**.

8.1.3.4 ProfileAccess.java

En esta clase se define la interfaz **ProfileAccess** del Servicio Web, siendo la encargada de ofrecer la interfaz *IdentifyPerson* del PIDS a través del Servicio Web. Esta interfaz posee tres métodos: ***get_traits_known***, ***get_profile*** y ***get_profile_list***.

8.1.3.5 ProfileAccessWS.java

Esta clase implementa la interfaz **ProfileAccess** del servicio web. La clase **ProfileAccessWS**, de forma análoga a *IdentifyPersonWS*, se encarga de envolver a la clase **ProfileAccessXML** (del paquete *Server.PID.Implementation*), implementada en el proyecto de partida y retocada ligeramente para este proyecto.

En esta clase se han implementado los tres métodos siguientes (los tres métodos reciben como argumento de entrada un OMElement y devuelven otro OMElement):

```
public OMElement get_traits_known (OMElement id)

public OMElement get_profile (OMElement entrada)

public OMElement get_profile_list (OMElement entrada)
```

El método ***get_traits_known*** recibe el identificador de una persona, necesario para poder invocar al método *get_traits_known* de la clase *ProfileAccessXML,* el cual devuelve el nombre de todos los atributos que se conocen de esa persona. Dichos atributos se recogen en el fichero ***get_traits_known_cod.xml*** que se crea en la carpeta *documentos/salidas.* El OMElement de salida del método *get_traits_known* de la clase *ProfileAccessWS* contiene el contenido de dicho fichero.

El método ***get_profile*** recibe en el OMElement de entrada el identificador de una persona más el contenido del fichero ***traits_requested_cod.xml*** (que se crea en la carpeta *documentos/entradas*), en el que se indican los nombres de los atributos solicitados. Con estos datos de entrada ya se puede invocar al método *get_profile* de la clase *ProfileAccessXML,* que devuelve el valor de todos los atributos solicitados para dicha persona a través del archivo ***profile_cod.xml***. El contenido del archivo *profile_cod.xml* es lo que se devuelve en el OMElement de salida.

Y por último, el método ***get_profile_list*** recibe en el OMElement de entrada una lista de identificadores de personas más el contenido del fichero ***traits_requested_cod.xml*** (que se crea en la carpeta *documentos/entradas* también), necesarios para poder invocar al método *get_profile_list* de la clase *ProfileAccessXML.* Este último método devuelve el valor de todos los atributos solicitados para todos los IDs que se hayan indicado a través del archivo ***taggedProfileSeq_*cod.xml**. El contenido del archivo *taggedProfileSeq_cod.xml* es lo que se devuelve en el OMElement de salida.

8.1.3.6 Auxiliar.java

La clase **Auxiliar** se utiliza tanto en servidor para realizar dos operaciones que se realizan en numerosas ocasiones en las clases *IdentifyPersonWS* y *ProfileAccessWS*. Esta clase posee dos métodos:

```
public static void crea_xml (String contenido_xml, String ruta_fichero)

public static OMElement get_OMElement (String ruta_fichero_xml)
```

El método ***crea_xml*** se encarga de crear un documento XML a partir del contenido que queramos que tenga dicho documento y la localización de éste (es decir, su ruta absoluta especificada mediante un *String*).

Y el método ***get_OMElement*** es el encargado de crear un OMElement a partir de un documento XML, para lo cual Axiom nos ha facilitado la labor. Simplemente conociendo la ruta del documento XML que se quiere transformar se puede obtener el OMElement correspondiente de la siguiente forma:

```
public static OMElement get_OMElement (String ruta_fichero_xml){

      OMElement salida=null;

      try {

            //create the parser

            XMLStreamReader parser = XMLInputFactory.newInstance()
            .createXMLStreamReader(new FileInputStream
            (ruta_fichero_xml));

            StAXOMBuilder builder= new StAXOMBuilder(parser);

            //get the root element
            salida= builder.getDocumentElement();

      } catch (XMLStreamException e) {
            e.printStackTrace();
            System.exit(1);

      } catch (FileNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
      }

      return salida;

}
```

Como se puede ver en el código del método ***get_OMElement***, y como ya se comentó en la introducción teórica, Axiom utiliza la tecnología StAX (*Streaming API for XML*), lo que hace que Axis2 tenga un mayor rendimiento que Axis1.

8.1.4 El Fichero de Configuración services.xml

Todo servicio web desplegado en Axis2 debe tener especificada su configuración en un fichero *services.xml,* el cual debe encontrarse en el directorio */META-INF* de la aplicación. Este fichero sirve para que Axis2 sepa, entre otras cosas, el nombre del servicio, el ámbito, los métodos que se implementan, los receptores de mensajes y el nombre de la clase principal que implementa el servicio.

El fichero *services.xml* que se ha usado en este proyecto es de la siguiente forma:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<serviceGroup>

<service name="IdentifyPerson">

     <description>
     This interface provides a way to identify a person (find a
     potencial ID) from the traits known about them.
     </description>

     <parameter name="ServiceClass"
     locked="false">Server.WebServiceClasses.IdentifyPersonWS
     </parameter>

     <operation name="find_candidates">
     <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
     class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
     <actionMapping>urn:find_candidates</actionMapping>
     </operation>

</service>


<service name="ProfileAccess">

     <description>
     This is the main interface for accessing the traits associated
     with an ID.
     </description>

     <parameter name="ServiceClass"
     locked="false">Server.WebServiceClasses.ProfileAccessWS
     </parameter>
```

```
    <operation name="get_profile">
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
    class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    <actionMapping>urn:get_profile</actionMapping>
    </operation>

    <operation name="get_profile_list">
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
    class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    <actionMapping>urn:get_profile_list</actionMapping>
    </operation>

    <operation name="get_traits_known">
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
    class="org.apache.axis2.receivers.RawXMLINOutMessageReceiver"/>
    <actionMapping>urn:get_traits_known</actionMapping>
    </operation>

</service>

</serviceGroup>
```

A continuación se comentan algunas de las etiquetas más importantes para que así resulte más fácil la comprensión del contenido del fichero *services.xml*:

- **service name**: El nombre del servicio será el nombre de la clase Java que define la interfaz del servicio.

- **description**: Este elemento es opcional, por si se quiere añadir una breve descripción del servicio.

- **parameter**: El fichero *services.xml* puede tener un gran número de parámetros, y todos los parámetros que se especifiquen serán transformados en características del servicio en la correspondiente descripción del servicio. El parámetro "ServiceClass" es el más importante y obligatorio, ya que especifica la clase java que realmente implementa el servicio, que será cargada en el *messageReceiver* (receptor de mensajes)

- **operations**: las operaciones son los métodos que van a ser publicados del servicio, y deben ser indicados en el fichero *services.xml*. El único atributo obligatorio de cada operación es el atributo *name (*el nombre del método).

8.1.5 Ficheros WSDL

Como ya comentamos en la introducción teórica, WSDL es el lenguaje XML que se utiliza para describir la interfaz publicada por un servicio web. Un documento WSDL proporciona toda la información necesaria para acceder y utilizar un servicio web: describe qué hace el servicio web, los tipos de datos que utiliza, cómo se comunica, y dónde reside.

Un documento WSDL utiliza los siguientes elementos en la definición de los servicios:

- **message**: Definición abstracta de los datos que se transmiten.
- **service**: Colección de puertos ("port").
- **port**: Descripción abstracta de una operación soportada por el servicio. Especifica una dirección para el enlace definiendo un único punto de destino.
- **port Type**: Conjunto de operaciones ("operations") soportadas por el servicio que definen el intercambio ordenado de mensajes. Con cada operación nos referimos a los mensajes de entrada y de salida.
- **binding**: Especifica el protocolo (SOAP, HTTP..) concreto que utiliza cada puerto y las especificaciones ("encoding") de los formatos de las operaciones.

Axis2 proporciona una herramienta para generar automáticamente los ficheros WSDL a partir de la clase en la que se define la interfaz del servicio. Esta herramienta se llama Java2WSDL (**org.apache.axis2.wsdl.java2wsdl.Java2WSDL**). En nuestro caso, a partir de las clases *IdentifyPerson* y *ProfileAccess* generamos los ficheros *IdentifyPerson.wsdl* y *ProfileAccess.wsdl*, respectivamente. Es muy importante para la generación de los ficheros WSDL que la variable de entorno *AXIS2_CLASSPATH* contenga todas las librerías necesarias.

8.2 Cliente

Para poder probar el correcto funcionamiento del Servicio Web para la Identificación de Personas, se ha desarrollado una aplicación cliente a modo de ejemplo. Evidentemente, dicha aplicación cliente no es la única manera de acceder al servicio web.

El nombre "cliente" puede resultar un tanto engañoso e inducir a error, ya que en realidad tiene dos facetas. Como cliente accede al Servicio Web del PIDS, y como servidor, sirve las peticiones que realizan los usuarios a través de un navegador web.

8.2.1 Estructura y Funcionamiento del Cliente

El cliente es una aplicación web, y como tal no tiene por qué estar instalado en la máquina del usuario. Que el cliente sea una aplicación web permite que pueda instalarse en cualquier máquina con un servidor de aplicaciones como Tomcat, y ser accedido vía web con cualquier navegador, haciéndolo fácilmente accesible para los usuarios. Por otro lado, es evidente que no es necesario tener el servidor y el cliente instalados en la misma máquina.

En nuestro caso, el cliente y el servidor se han instalado en la misma máquina. Sin embargo, Axis2, y por tanto el servidor, se desplegaron en un Tomcat que escuchaba en el puerto 8080 (el puerto por defecto), y la aplicación web cliente fue desplegada en otro Tomcat que escuchaba por el puerto 18080.

El funcionamiento del cliente en líneas generales es el siguiente:

- El usuario se conecta mediante un navegador web al cliente.

- El cliente devuelve una página HTML (*index.html*) donde se indica la funcionalidad de las dos interfaces del PIDS que se implementan (*IdentifyPerson* y *ProfileAccess*). El usuario debe elegir una de las interfaces seleccionando el enlace correspondiente.

- Una vez que el usuario ha elegido una de las dos interfaces pasará a la

página HTML, *IdentifyPerson.html* o *ProfileAccess.html*, según la elección. Y es en una de estas dos páginas donde elegirá uno de los métodos que se implementan de cada interfaz, según la consulta que quiera realizar. En el diagrama 8.3 se muestra el Diagrama de Casos de Uso de la aplicación.

- Por último el usuario rellena un formulario con la información necesaria para realizar la consulta elegida.

- Dicha información se manda al *servlet* (la clase C**lient** de la aplicación web).

- El *servlet* procesa la información a través de la clase **Create_XML** y se la pasa a la clase **IdentifyPersonClient** o a **ProfileAccessClient**, según convenga.

- Estas dos últimas clases, usando Axis2, mandan el mensaje SOAP de petición al Servicio Web y reciben el mensaje SOAP de respuesta de éste. Después extraen la información de respuesta y se la pasan al *servlet*.

- El *servlet*, por último, envía en una página HTML el resultado de la petición.

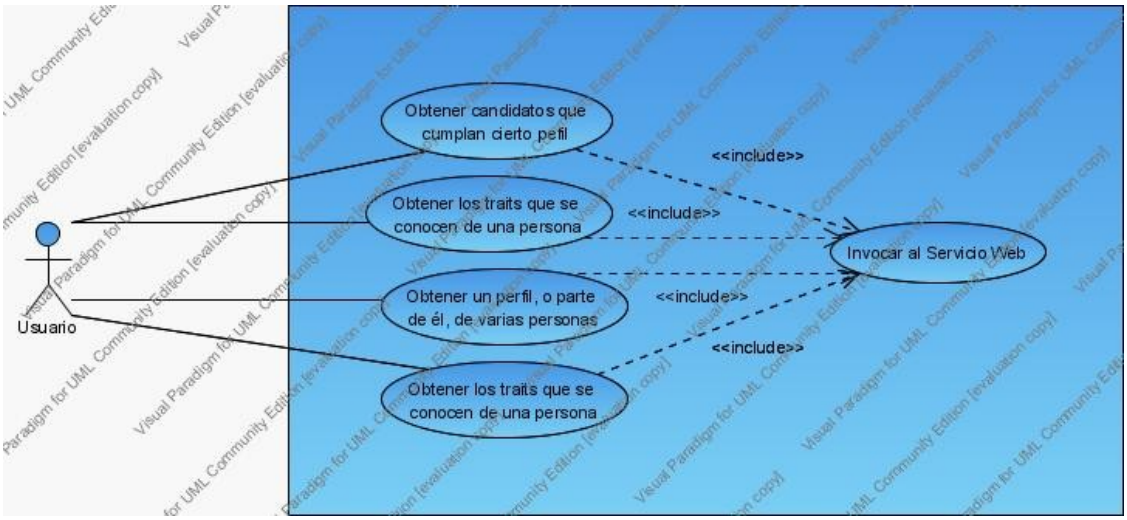

Diagrama 8.3: Diagrama de Casos de Uso de la aplicación

Comentar que el cliente, antes de enviar una petición al servidor, canoniza los documentos XML de entrada de tal forma que no contengan ni acentos ni caracteres 'ñ'. Y luego realiza el proceso contrario, recibe del servidor documentos XML canonizados, los procesa y se los muestra al usuario en el formato adecuado.

El cliente, como aplicación web que es, presenta la estructura que se muestra en la figura 8.3:

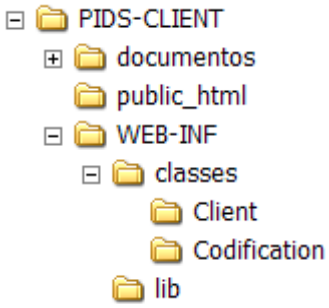

Figura 8.3: Estructura de directorios del Cliente

- En el directorio origen de la aplicación se encuentran las páginas HTML.
- En **/documentos** se almacenan todos los documentos XML que se van creando.
- La clase *servlet* (*Client.java*), debe montarse en un servidor de aplicaciones (en nuestro caso Tomcat), en el directorio **/WEB-INF/classes/Client**.
- Las clases de las que se sirve el *servlet* para realizar su tarea se encuentran también en el directorio **/WEB-INF/classes/Client**.
- El archivo descriptor de despliegue de la aplicación web, *web.xml*, debe estar en el directorio **/WEB-INF**.
- En **/WEB-INF/classes/Codification**, se encuentra la clase *Canonizador.java* (también usada en el servidor) encargada de canonizar los

documentos XML para que a priori no contengan caracteres no válidos como pudieran ser las 'ñ' o las tildes.

- Y por último comentar que para las transformaciones Java-XML mediante Castor son necesarios los mapas que encontraremos en **/public_html**, así como que el paquete **PID.DataTypes** se encuentre en el directorio *$CATALINA_HOME/common/classes* del contenedor de aplicaciones donde se encuentre desplegada la aplicación web cliente.

8.2.2 Diagrama de clases

A continuación se muestra el diagrama de clases del paquete **Client**, donde se encuentra el *servlet* y el resto de las clases java que éste utiliza para recopilar la información necesaria para preparar la petición e invocar al servidor.

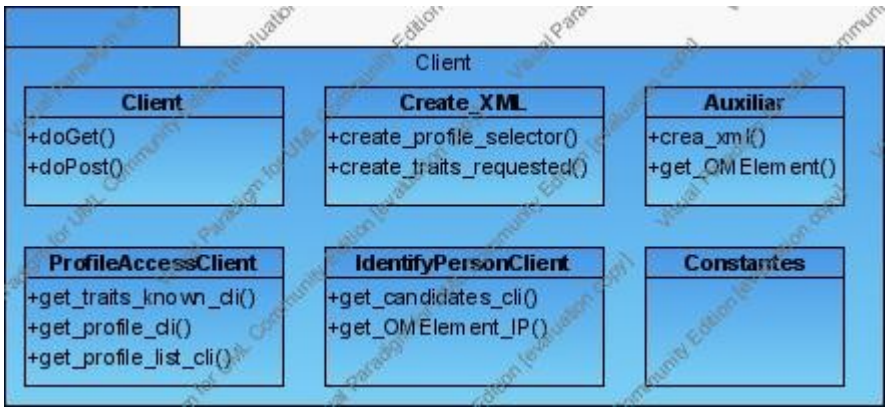

Diagrama 8.3: Diagrama de Clases del paquete Client

8.2.3 Clases Principales

En este apartado se comentan las principales características de las clases java que componen la aplicación web cliente.

8.2.3.1 Client.java

La clase **Client** es un *servlet* que es invocado por las páginas HTML en las que se recogen la información necesaria para invocar a cada uno de los métodos de las interfaces: *FindCandidates.html*, *GetTraitsKnown.html*, *GetProfile.html* y *GetProfileList.html*. Al ser un *servlet* recibe como parámetros los mensajes de petición y respuesta HTTP. Del mensaje de petición extrae la información necesaria para invocar al Servicio Web, y con el mensaje de respuesta devuelve el resultado al usuario (en realidad a su navegador).

El *servlet* para realizar su labor se sirve de otras clases de la aplicación web:

- Utiliza **Create_XML** para crear los documentos XML de entrada *traits_requested*.xml y *profile_selector*.xml, según convenga.

- Utiliza **IdentifyPersonClient** para invocar al método *find_candidates* de la interfaz *IdentifyPerson* del Servicio Web.

- Utiliza **ProfileAccessClient** para invocar a los métodos implementados de la interfaz *ProfileAccess* del Servicio Web.

Cuando se obtienen los resultados de la petición, el *servlet* envía el resultado al navegador del usuario remoto mediante el mensaje de respuesta HTTP.

8.2.3.2 Create_XML

Esta clase es la encargada de crear los documentos XML de entrada para el método del PIDS que lo requiera. Posee dos métodos: ***create_profile_selector*** y ***create_traits_requested***. Ambos reciben el mensaje de petición HTTP, a partir del cual recopilan la información necesaria para crear el documento *profile_selector.xml* y *traits_requested.xml*, respectivamente, devolviendo un *String* con la ruta del documento XML creado. Para crear los documentos XML a partir de objetos Java se ha utilizado, al igual que en el servidor, el *framework* Castor.

```
public String create_profile_selector ( HttpServletRequest request );

public String create_traits_requested ( HttpServletRequest request );
```

8.2.3.3 IdentifyPersonClient.java y ProfileAccessClient.java

Estas dos clases se encargan de todo lo relacionado con los servicios web, SOAP, Axis2, Axiom, ocultándoselo al *servlet*. Reciben toda la información que necesitan para que con la ayuda de Axiom se cree el OMElement de petición y posteriormente invocar al Servicio Web.

La clase **IdentifyPersonClient** posee un sólo método, *find_candidates_cli*, y la clase **ProfileAccessClient** tres métodos: *get_traits_known_cli*, *get_profile_cli* y *get_profile_list_cli*. Las declaraciones de dichos métodos son las siguientes:

```
public String find_candidates_cli(String ruta_profile_selector,
IdStateSeq states_of_interest, float confidence_threshold, long
sequence_max, long iterator_max, String ruta_traits_requested)

public String get_traits_known_cli (String id)

public String get_profile_cli (String id, String
ruta_traits_requested)

public String get_profile_list_cli (String id [], String
ruta_traits_requested);
```

El método ***find_candidates_cli*** es el encargado de realizar todas las operaciones necesarias para poder invocar al método ***find_candidates*** de la interfaz *IdentifyPerson* del servicio web. El método ***get_traits_known_cli*** se encarga de preparar todo lo necesario para poder invocar al método ***get_traits_known*** de la interfaz *ProfileAccess* del servicio web, y así de forma análoga para los otros dos métodos.

Los cuatro métodos reciben todos los parámetros que necesitan para invocar al método correspondiente del servicio web. El método ***find_candidates_cli***, ***get_profile_cli*** y ***get_profile_list_*****cli**, que tienen como parámetro de entrada algún documento XML, mejor dicho la ruta absoluta de algún documento XML almacenada en un *String,* lo primero que realizan es la canonización del documento XML de entrada (o los documentos XML en el caso del método ***find_candidates_cli***) para enviarlos sin acentos ni caracteres 'ñ', como ya se ha comentado anteriormente (haciendo uso de la clase **Canonizador**, que encontraremos en el paquete *Codification*). A continuación preparan el OMElement de petición, invocan al servicio web, y recuperan el documento, o los documentos, XML de salida del OMElement que reciben del servidor.

8.2.4 El archivo descriptor de despliegue: web.xml

El fichero *web.xml,* el cual debe encontrarse obligatoriamente en el directorio */WEB-INF* de la aplicación web, es el descriptor de despliegue de la aplicación, donde se indica principalmente los servlets que utilizará la aplicación. En nuestro caso la aplicación sólo tiene un *servlet*, la clase **Client** (que se encuentra dentro del paquete *Client*).

A continuación se muestra el contenido del fichero *web.xml* que se ha utilizado en nuestra aplicación web:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

      <display-name>PIDSClient</display-name>

      <servlet>
      <description>Servlet principal</description>
      <servlet-name>Client</servlet-name>
      <servlet-class>Client.Client</servlet-class>
      </servlet>
```

```
        <servlet-mapping>
        <servlet-name>Client</servlet-name>
        <url-pattern>/Client</url-pattern>
        </servlet-mapping>

</web-app>
```

8.3 Funcionamiento Global

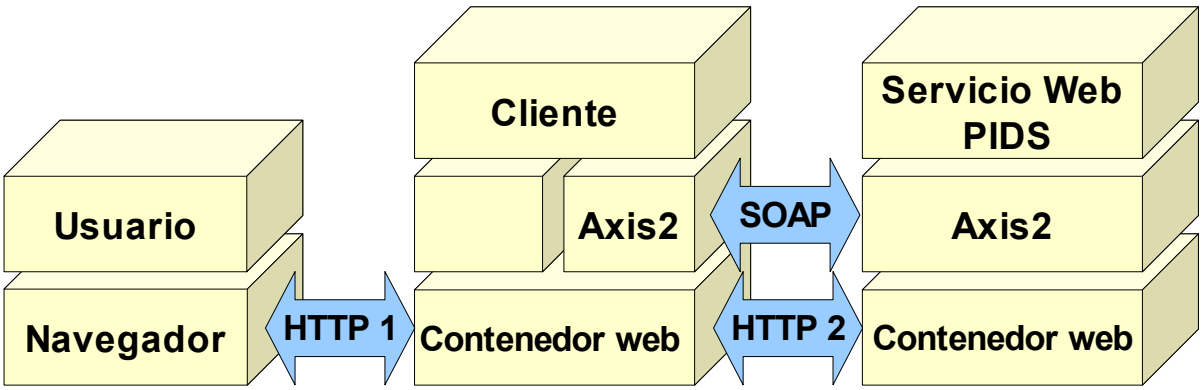

Figura 8.4: Esquema sobre el funcionamiento global de la aplicación

Tal y como se muestra en la figura 8.4, el funcionamiento de la aplicación en líneas generales se puede resumir de la siguiente forma:

1. El usuario accede a la página web donde está ubicado el cliente, elige alguno de los métodos implementados de las interfaces *IdentifyPerson* y *ProfileAccess* según la consulta que desee hacer, y rellena un formulario con los datos necesarios para dicha consulta, mandándose una petición HTTP.

2. El contenedor web (en nuestro caso Tomcat) invoca el cliente.

3. El cliente obtiene los datos que se introdujeron en el formulario y utiliza Axis2 para llamar al servidor.

4. Axis2 genera el mensaje SOAP de petición y lo empaqueta en una petición

HTTP (que identificamos como 2 en la figura 8.4) que manda al servidor (aunque, como ya hemos comentado anteriormente, podría utilizarse otro protocolo distinto de HTTP).

5. El contenedor web del servidor recibe la petición HTTP y decide mandársela a Axis2.

6. Axis2 decide invocar al método del PIDS que se haya elegido y obtiene los resultados.

7. Axis2 construye el mensaje SOAP de respuesta y lo manda hacia el cliente en un mensaje de respuesta HTTP mediante el contenedor web.

8. El Axis2 del cliente recibe la respuesta HTTP y la desempaqueta obteniendo la respuesta SOAP, que también desempaqueta obteniendo el resultado de la consulta.

9. El Axis2 del cliente devuelve en la respuesta HTTP el resultado de la petición, que recibe el navegador para que el usuario pueda visualizarlo.

PARTE IV : CONCLUSIONES Y FUTURAS AMPLIACIONES

9 Conclusiones

El objetivo de este proyecto era desarrollar una aplicación que permitiese el fácil y rápido acceso a un sistema de información que almacena información demográfica completa de los ciudadanos. Siendo uno de los requisitos indispensables el cumplimiento de la especificación del Servicio de Identificación de Personas de CORBAmed, y que el acceso a toda esta información se realizara haciendo uso de la tecnología de Servicios Web. Este objetivo se ha cumplido utilizando el proyecto de Apache Axis2 (desplegado sobre el contenedor de aplicaciones Tomcat) y reutilizando y ampliando algunos trabajos previos en esta línea.

Al final se ha conseguido un Servicio Web con las siguientes características:

- Permite tener acceso a la información demográfica almacenada a través del Servicio de Identificación de Personas (PIDS). Es decir, nos permite identificar de forma unívoca a una persona dentro de un colectivo a partir de ciertas características distintivas de la misma, así como extraer información demográfica y administrativa de una persona partiendo de su identificador.

- Es independiente de la plataforma, por estar construido en Java. Comentar que, aunque se ha tomado otra aplicación como base para algunas funciones y se ha podido reutilizar código, ha sido necesario un gran estudio de Java.

- Es fácilmente accesible y reutilizable. Gracias a la tecnología de Servicios Web resulta muy fácil implementar clientes que accedan al servicio.

- Es independiente del protocolo de transporte, pudiendo ser éste HTTP o cualquier otro, al ser un servicio web que utiliza SOAP. Lo que junto a las dos características anteriores le proporciona gran flexibilidad.

- Utiliza software de código abierto.

Además, para probar el correcto funcionamiento del servicio web implementado, se ha desarrollado una interfaz web capaz de invocar el servicio pasándole una serie de parámetro mediante el uso de *servlets*.

10 Futuras ampliaciones

En este apartado de la memoria se van a comentar las posibles ampliaciones o mejoras que se podrían realizar de forma que se consiga una mayor funcionalidad, mejorar su seguridad o hacerlo más eficiente en cuanto a reducción de código o tiempo de procesado.

Algunas de esas posibles mejoras son:

- Completar la librería de tipos CEN. Como se comentó en la introducción teórica, el estándar *Health informatics-prEN 14822-2:2003* se dividía en tres partes, de las cuales sólo los elementos de las dos primeras partes se implementan en el proyecto. Por ello, añadiendo la tercera parte denominada ***General purpose components for messages – Clinical***, que contiene una serie de especificaciones sobre datos clínicos, se completaría la librería de tipos CEN.

- Implementar otras interfaces del PIDS. En el servidor de partida se implementaban la interfaz *IdentifyPerson* y parte de la interfaz *ProfileAccess*, que son las que se han ofrecido a través del servicio web implementado, pero podrían implementarse otras como la interfaz **IdMgr** o la interfaz **CorrelationMgr**. La interfaz *IdMgr* proporciona funciones para la gestión de identificadores en un Dominio de Identificación y la interfaz *CorrelationMgr* permitiría identificar personas utilizando los datos almacenados en varios dominios, además de la extensión de la interfaz para la gestión de más de un Dominio de Identificación.

- Cifrar los datos. Sería bueno garantizar la confidencialidad de los datos intercambiados, ya que SOAP no cifra la información, que viaja en claro a través de Internet. El estándar ya firmemente establecido de creación de canales seguros SSL y el cifrado de partes específicas de documentos mediante el cifrado XML son las direcciones que se están siguiendo en este terreno.

- Mejoras en la aplicación web cliente, como por ejemplo todas las derivadas del hecho de ampliar el número de interfaces implementadas en el servidor, así como ampliar el número de atributos selectores que ayudan a obtener los posibles

candidatos. Otra posible ampliación podría ser presentar de una forma más elaborada los resultados al usuario remoto en el navegador. Son muchos los cambios que se podrían hacer en la aplicación web cliente.

PARTE V : MANUAL DE USUARIO

11 Manual de Usuario

11.1 Introducción

En este apartado de la memoria se van a explicar los pasos que hay que ir realizando para poner en funcionamiento el Servicio Web y probar su funcionamiento a través de la interfaz web implementada.

En nuestro caso, tanto el cliente como el servidor se han instalado en la misma máquina. Pero Axis2, y por tanto el servidor, se desplegaron en un Tomcat que escuchaba en el puerto 8080 (el puerto por defecto), y la aplicación web cliente se desplegó en otro Tomcat, que escuchaba por el puerto 18080.

Se va a suponer para los apartados posteriores que ya se tiene instalado el JDK, el contenedor de aplicaciones Tomcat, MySQL y Axis2 está desplegado correctamente sobre Tomcat. Y, obviamente, que se ha iniciado MySQL y Tomcat.

11.2 Creación de la Base de Datos

El primer paso es la creación de la base de datos MySQL. Para ello, se deben realizar los siguientes pasos:

1. Crear una base de datos vacía llamada *PersonDB*.

2. Crear un usuario para la creación y manejo de dicha base de datos (*PersonDB*). En nuestro caso se creó el usuario "userpids" con contraseña "passwordpids". En caso de que se desee crear otro distinto hay que actualizar la clase **DB_Manager.java** del paquete **Server.Local.PID.DataBase**.

3. Incluir el conector *mysql-connector-java-5.0.4-bin.jar* en el CLASSPATH.

4. Ejecutar la clase **Crea_DB.java** que se encarga de crear la base de datos.

5. Para comprobar que se ha creado correctamente la base de datos podemos

hacerlo a través del gestor de base de datos phpMyAdmin (previamente iniciado el servidor Apache), accediendo mediante la URL http://localhost/phpMyAdmin con el usuario **root**.

11.3 Librerías necesarias

Los tipos de datos se encontraban implementados como un archivo de extensión .jar, llamado *Librería.jar*, para permitir la incorporación a otros proyectos que usen el mismo modelo de datos.

Para permitir la conexión con la base de datos MySQL creada y poder realizar consultas es necesario incluir el conector *mysql-connector-java-5.0.4-bin.jar*

Debido al uso de código XML y del *framework* Castor es necesario incluir las siguientes librerías en el proyecto:

- common-logging-1.1.jar
- castor.jar
- xalan-2.0.1.jar
- xerces.1.4.3.jar

Las dos primeras librerías permiten el uso de Castor, y las dos siguientes son necesarias para que Castor lleve a cabo sus funciones.

Para poder usar Axiom resulta necesario incluir también las siguientes librerías en el proyecto:

- axiom-api-1.2.4.jar
- axiom-dom.1.2.4.jar

- axiom.impl-1.2.4.jar

El resto de librerías necesarias para la aplicación se encontrarán en el directorio *$CATALINA_HOME/comnmon/lib* de Tomcat y en el directorio *$AXIS2_HOME/WEB_INF/lib* de Axis2 una vez desplegado.

11.4 Despliegue del Servicio Web

Para el despliegue del Servicio Web implementado sobre Axis2 se ha creado un script llamado **crea_service.sh**. Este script te lleva al directorio origen del la estructura del servidor (*PIDS-WS*), crea el archivo **MyWebService.aar** y lo copia en el directorio *$AXIS2_HOME/WEB-INF/services* de Axis2. El script *crea_service.sh* lo podemos encontrar en el directorio *PIDS-WS* (directorio raíz de la estructura del servidor).

No olvidar comprobar que en el archivo *$AXIS2_HOME/WEB-INF/services/*services.list aparezca una entrada con el nombre del archivo .aar. Y por último, comentar que para que la transformación Java-XML funcione es necesario colocar el paquete **PID.DataTypes** en *$AXIS_HOME/WEB-INF/classes*.

Para comprobar que el Servicio Web se ha desplegado correctamente, podemos acceder a la URL http://localhost:8080/axis2/services/listServices y verificar que aparecen las dos interfaces implementadas (*IdentifyPerson* y *ProfileAccess*), tal y como se muestra en la captura de pantalla mostrada en la figura 11.1. Para cada interfaz se indican las operaciones (o métodos) que se publican en el servicio, y si seleccionamos cada una de las interfaces nos aparecerá el archivo WSDL correspondiente a dicha interfaz.

Available services

IdentifyPerson

Service EPR : http://localhost:8080/axis2/services/IdentifyPerson
Service REST epr : http://localhost:8080/axis2/rest/IdentifyPerson

Service Description : This interface provides a way to identify a person (find a potencial ID) from the traits known about them.

Service Status : Active
Available Operations

- find_candidates

ProfileAccess

Service EPR : http://localhost:8080/axis2/services/ProfileAccess
Service REST epr : http://localhost:8080/axis2/rest/ProfileAccess

Service Description : This is the main interface for accessing the traits associated with an ID.

Service Status : Active
Available Operations

- get_traits_known
- get_profile_list
- get_profile

Version

Service EPR : http://localhost:8080/axis2/services/Version
Service REST epr : http://localhost:8080/axis2/rest/Version

Service Description : This service is to get the running Axis version

Service Status : Active
Available Operations

- getVersion

Figura 11.1: Servicios desplegados en Axis2

11.5 Despliegue del Cliente

El cliente, al ser una aplicación web, debe instalarse en un contenedor de aplicaciones. En nuestro caso vamos a usar también Tomcat, escuchando en el puerto 18080. El despliegue de la aplicación web consiste en copiar el archivo **PIDSClient.war** en el directorio *$CATALINA_HOME/webapps*.

Comentar que el cliente necesita para las transformaciones Java-XML mediante el uso de Castor de los mapas que encontraremos en **/public-html**. Además, es necesario colocar el paquete **PID.DataTypes** (el mismo del servidor) en el directorio *$CATALINA_HOME/common/classes* para que Castor pueda realizar correctamente su labor.

Una vez que se ha desplegado la aplicación web sobre Tomcat, a través de la URL http://localhost:18080/PIDSClient/index.html (siempre y cuando se acceda desde la misma máquina en la que se haya desplegado la aplicación, en caso contrario habría que cambiar *localhost* por la dirección IP que corresponda) ya podríamos probar el funcionamiento de la aplicación.

11.6 Estructura Completa de la Distribución

El directorio raíz de la distribución es **/PFC**, el cual presenta la estructura de directorios que se observa en la figura 11.2, cuyo contenido se indica a continuación:

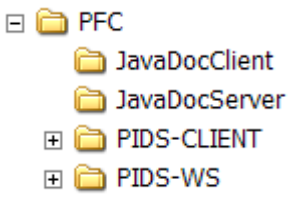

Figura 11.2: Estructura de completa de la distribución

- **JavaDocClient:** JavaDoc correspondiente al cliente.

- **JavaDocServer**: JavaDoc correspondiente al servidor.

- **PIDS-CLIENT**: Es la aplicación web cliente. Cumple con la estructura de directorios típica de las aplicaciones web, donde las clases java están en el directorio **/WEB-INF/classes**, las librerías en **/WEB-INF/lib**, las páginas HTML en el directorio raíz (**/PIDS-CLIENT**), y el descriptor de despliegue (*web.xml*) en **/WEB-INF**. En **/documentos/salidas** encontraremos los documentos XML con los resultados de las consultas realizadas. Además, encontraremos el archivo *PIDSClient.war* correspondiente a la aplicación web.

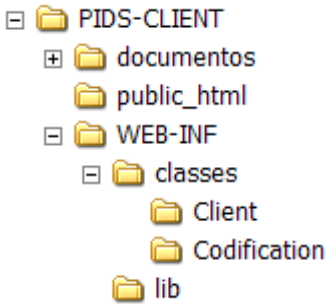

Figura 11.3: Estructura de directorios del Cliente

- **PIDS-WS**: Contiene todo lo relacionado con el servidor, tal y como se muestra en la figura 11.4. En **/documentos** se almacenan los documentos de entrada y de salida del código, en **/lib** las librerías que utiliza el servidor y en **/public_html** se almacenan los mapas necesarios para las transformaciones Java-XML mediante el uso de Castor. En el directorio **/META-INF** encontramos el archivo descriptor del servicio (*services.xml*) y los archivos *IdentifyPerson.wsdl* y *ProfileAccess.wsdl,* y en **/Server** todas las clases java que implementan el servidor. Además, en /PIDS-WS encontramos el fichero *Crea_DB.java*, que se encarga de crear la base de datos, y el script *crea_service.sh*, encargado de desplegar el servicio web en Axis2.

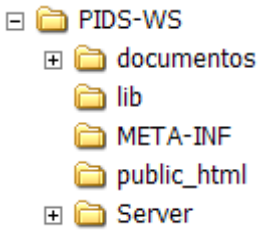

Figura 11.4: Estructura de directorios del Servidor

PARTE VI: BIBLIOGRAFÍA

12 Bibliografía

- Estándar "*Health informatics – Data types"* del Comité Europeo de Normalización (CEN).

- Estándar "*Health informatics – General purpose information components"* del Comité Europeo de Normalización (CEN).

- Especificación *"Person Identification Service (PIDS) Specification"* de CORBAmed.

- "Manual de referencia de MySQL 5.0".

- "De la historia clínica a la historia de salud electrónica" [en línea]. http://www.conganat.org/seis/informes/2003/

- "Por qué XML?" [en línea]. http://www.gamarod.com.ar/articulos/por_que_xml.asp

- "Arquitectura cliente-servidor" [en línea] http://www.csi.map.es/csi/silice/Global71.html

- Documentación de Axis2 [en línea]. http://ws.apache.org/axis2/1_2/contents.html

- "Axi2" [en línea] http://ame.endesa.es/confluence/display/AMEBASE/AXIS2

- Documentación de Axiom [en línea]. http://ws.apache.org/commons/axiom/OMTutorial.html

- Documentación de Castor [en línea]. http://www.castor.org

- "The Java Tutorial" [en línea]. http://java.sun.com/docs/books/tutorial/

- "Eclipse Tutorial" [en línea]. http://eclipsetutorial.forge.os4os.org/in1.htm

- "Servlet Essentials" [en línea] http://www.novocode.com/doc/servlet-essentials

- Repullo López, María Ángeles. "Servidor demográfico basado en normas del CEN y CORBAmed".

- Caminero Marqués, Mª José. "Pasarela Java-XML para GPICS no clínicos y aplicación en un servidor demográfico".

PARTE VII: DOCUMENTACIÓN DE LA APLICACIÓN

13 Documentación de la Aplicación

A continuación se presenta el JavaDoc de la aplicación, el cual ha sido generado a partir de los comentarios que se han ido añadiendo al código fuente durante el desarrollo de la aplicación.

13.1 Servidor

Paquetes de la Aplicación

Packages	
Server.Local.PID.DataBase	Este paquete contiene las clases necesarias para que el servicio pueda interactuar con la base de datos y realizar búsquedas de atributos e identificadores de persona.
Server.Local.PID.ServiceClasses	En este paquete encontramos las clases necesarias para que el servicio pueda acceder a los métodos que implementan las interfaces IdentifyPerson y ProfileAccess del Sistema de Identificación de Personas (PIDS) de CORBAmed (find_candidates, get_traits_known, get_profile y get_profile_list), a partir de las cuales se realizan las consultas a la base de datos para obtener identificadores de persona o valores de atributos.
Server.PID.DataTypes	Este paquete contiene las clases que encapsulan a los tipos de datos definidos en el PIDS en el apartado de Data Types.
Server.PID.Exception	Este paquete contiene las clases que encapsulan a las excepciones definidas en el PIDS en el apartado de Data Types.
Server.PID.Implementation	Este paquete contiene las clases necesarias para que el servicio pueda realizar sus funciones de pasarela java-XML.
Server.WebServiceClasses	En este paquete están todas las clases que ofrecen las interfaces del PIDS a través del Servicio Web.

Figura 13.1: Paquetes del Servidor.

Paquete Server.WebServiceClasses

Interface Summary	
IdentifyPerson	Clase en la que se define la interfaz IdentifyPerson del Servicio Web.
ProfileAccess	Clase en la que se define la interfaz ProfileAccess del Servicio Web.

Class Summary	
Auxiliar	Clase donde se implementan los métodos crea_xml y get_OMELement.
Constantes2	Clase donde se definen las constantes con las rutas de los documentos XML que se crean en el servidor.
IdentifyPersonWS	Clase que implementa la interfaz IdentifyPerson del Servicio Web.
ProfileAccessWS	Clase que implementa la interfaz ProfileAccess del Servicio Web.

Figura 13.2: Paquete Server.WebServiceClasses.

Server.WebServiceClasses
Interface IdentifyPerson

All Known Implementing Classes:
IdentifyPersonWS

---

public interface **IdentifyPerson**

Clase en la que se define la interfaz IdentifyPerson del Servicio Web.

Author:
Pilar Mª Juan Montilla

Method Summary	
org.apache.axiom.om.OMElement	**find_candidates**(org.apache.axiom.om.OMElement peticion) Método que recibe todos los parámetros necesarios para invocar al método find_candidates de la clase IdentifyPersonXML.

Method Detail

find_candidates

`org.apache.axiom.om.OMElement` **`find_candidates`**`(org.apache.axiom.om.OMElement peticion)`

Método que recibe todos los parámetros necesarios para invocar al método find_candidates de la clase IdentifyPersonXML.

Parameters:

`peticion` - OMElement con todos los parámetros de entrada.

Returns:

OMElement de respuesta con los documentos de salida returned_sequence_cod.xml y returned_iterator_cod.xml.

OMElement de respuesta con los documentos de salida returned_sequence_cod.xml y returned_iterator_cod.xml.

`Server.WebServiceClasses`
`Interface ProfileAccess`

All Known Implementing Classes:

ProfileAccessWS

`public interface` **`ProfileAccess`**

Clase en la que se define la interfaz ProfileAccess del Servicio Web.

Author:

Pilar Mª Juan Montilla

Method Summary

`org.apache.axiom.om.OMElement`	**`get_profile_list`**`(org.apache.axiom.om.OMElement entrada)` Método que recibe todos los parámetros necesarios para invocar al método get_profile_list de la clase ProfileAccessXML.
`org.apache.axiom.om.OMElement`	**`get_profile`**`(org.apache.axiom.om.OMEl`

	ement entrada) Método que recibe todos los parámetros necesarios para invocar al método get_profile de la clase ProfileAccessXML.
org.apache.axiom.om.OMElement	**get_traits_known**(org.apache.axiom.om.OMElement id) Método que recibe todos los parámetros necesarios para invocar al método get_traits_known de la clase ProfileAccessXML.

Method Detail

get_profile

```
org.apache.axiom.om.OMElement get_profile(org.apache.axiom.om.OMElement
entrada)
```

Método que recibe todos los parámetros necesarios para invocar al método get_profile de la clase ProfileAccessXML.

Parameters:

entrada - OMElement con todos los parámetros de entrada: uno identificador de persona y el contenido del documento de entrada traits_requested_cod.xml

Returns:

OMElement de respuesta con el contenido del fichero de salida profile_cod.xml

---

get_profile_list

```
org.apache.axiom.om.OMElement
get_profile_list(org.apache.axiom.om.OMElement entrada)
```

Método que recibe todos los parámetros necesarios para invocar al método get_profile_list de la clase ProfileAccessXML.

Parameters:

entrada - OMElement con todos los parámetros de entrada: uno o varios identificadores de persona y el contenido del documento de entrada traits_requested.xml

Returns:

OMElement de respuesta con el contenido del fichero de salida taggedProfileSeq_cod.xml

---

get_traits_known

```
org.apache.axiom.om.OMElement
get_traits_known(org.apache.axiom.om.OMElement id)
```

Método que recibe todos los parámetros necesarios para invocar al método get_traits_known de la clase ProfileAccessXML.

Parameters:

`id` - OMElement con un identificador de persona.

Returns:

OMElement de respuesta con el contenido del fichero de salida traits_known_cod.xml

---

Server.WebServiceClasses
Class Auxiliar

```
java.lang.Object
  Server.WebServiceClasses.Auxiliar
```

```
public class Auxiliar
extends java.lang.Object
```

Clase donde se implementan los métodos crea_xml y get_OMELement.

Author:

Pilar Mª Juan Montilla

Constructor Summary

Auxiliar()	

Method Summary

static void	**crea_xml**(java.lang.String contenido_xml, java.lang.String ruta_fichero) Método que crea un fichero XML a partir de su contenido y su ruta absoluta.
static org.apache.axiom.om.OMElement	**get_OMElement**(java.lang.String ruta_fichero_xml) Método que obtiene el OMElement

	correspondiente a un fichero XML.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Auxiliar

```
public Auxiliar()
```

Method Detail

crea_xml

```
public static void crea_xml(java.lang.String contenido_xml,
                            java.lang.String ruta_fichero)
```

Método que crea un fichero XML a partir de su contenido y su ruta absoluta.

Parameters:

`contenido_xml` - String con el contenido del fichero XML a crear.
`ruta_fichero` - String con la ruta del fichero XML a crear.

---

get_OMElement

```
public static org.apache.axiom.om.OMElement get_OMElement(java.lang.String
ruta_fichero_xml)
```

Método que obtiene el OMElement correspondiente a un fichero XML.

Parameters:

`ruta_fichero_xml` - String que contiene la ruta del fichero XML en cuestión.

Returns:

El OMELement de salida

Server.WebServiceClasses
Class Constantes2
java.lang.Object
  Server.WebServiceClasses.Constantes2

---

public class **Constantes2**extends java.lang.Object

Clase donde se definen las constantes con las rutas de los documentos XML que se crean en el servidor.

Author:
Pilar Mª Juan Montilla

Field Summary	
static java.lang.String	**PIDS** Ruta del mapa PIDs.xml, necesario para la transformación Java-XML mediante el uso de Castor
static java.lang.String	**PROFILE_COD** Ruta donde se crea el fichero profile_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
static java.lang.String	**PROFILE_DECO** Ruta donde se crea el fichero profile_deco.xml
static java.lang.String	**PROFILE_SELECTOR_COD** Ruta donde se crea el fichero profile_selector_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
static java.lang.String	**PROFILE_SELECTOR_DECO** Ruta donde se crea el fichero profile_selector_deco.xml
static java.lang.String	**RETURNED_ITERATOR_COD** Ruta donde se crea el fichero returned_iterator_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
static java.lang.String	**RETURNED_ITERATOR_DECO** Ruta donde se crea el fichero returned_iterator_deco.xml
static java.lang.String	**RETURNED_SEQUENCE_COD** Ruta donde se crea el fichero returned_sequence_cod.xml ("cod" de codificado, es decir, sin acentos)
static java.lang.String	**RETURNED_SEQUENCE_DECO** Ruta donde se crea el fichero returned_sequence_deco.xml
static java.lang.String	**TAGGEDPROFILESEQ_COD**

	Ruta donde se crea el fichero taggedProfileSeq_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
`static java.lang.String`	**`TAGGEDPROFILESEQ_DECO`** Ruta donde se crea el fichero taggedProfileSeq_deco.xml
`static java.lang.String`	**`TRAITS_KNOWN_COD`** Ruta donde se crea el fichero traits_known_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
`static java.lang.String`	**`TRAITS_KNOWN_DECO`** Ruta donde se crea el fichero traits_known_deco.xml
`static java.lang.String`	**`TRAITS_REQUESTED_COD`** Ruta donde se crea el fichero traits_requested_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
`Static java.lang.String`	**`TRAITS_REQUESTED_DECO`** Ruta donde se crea el fichero traits_requested_deco.xml

Constructor Summary

`Constantes2``()`	

Method Summary

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

PROFILE_SELECTOR_COD

`public static final java.lang.String` **`PROFILE_SELECTOR_COD`**

Ruta donde se crea el fichero profile_selector_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

PROFILE_SELECTOR_DECO

`public static final java.lang.String` **`PROFILE_SELECTOR_DECO`**

Ruta donde se crea el fichero profile_selector_deco.xml

See Also:
Constant Field Values

TRAITS_REQUESTED_COD

`public static final java.lang.String` **`TRAITS_REQUESTED_COD`**

Ruta donde se crea el fichero traits_requested_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

TRAITS_REQUESTED_DECO

`public static final java.lang.String` **`TRAITS_REQUESTED_DECO`**

Ruta donde se crea el fichero traits_requested_deco.xml

See Also:
Constant Field Values

RETURNED_SEQUENCE_COD

`public static final java.lang.String` **`RETURNED_SEQUENCE_COD`**

Ruta donde se crea el fichero returned_sequence_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

RETURNED_SEQUENCE_DECO

`public static final java.lang.String` **`RETURNED_SEQUENCE_DECO`**

Ruta donde se crea el fichero returned_sequence_deco.xml

See Also:
Constant Field Values

RETURNED_ITERATOR_COD

```
public static final java.lang.String RETURNED_ITERATOR_COD
```

Ruta donde se crea el fichero returned_iterator_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

RETURNED_ITERATOR_DECO

```
public static final java.lang.String RETURNED_ITERATOR_DECO
```

Ruta donde se crea el fichero returned_iterator_deco.xml

See Also:
Constant Field Values

PIDS

```
public static final java.lang.String PIDS
```

Ruta del mapa PIDs.xml, necesario para la transformación Java-XML mediante el uso de Castor

See Also:
Constant Field Values

PROFILE_COD

```
public static final java.lang.String PROFILE_COD
```

Ruta donde se crea el fichero profile_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:

Constant Field Values

PROFILE_DECO

```
public static final java.lang.String PROFILE_DECO
```

Ruta donde se crea el fichero profile_deco.xml

See Also:
Constant Field Values

TAGGEDPROFILESEQ_COD

```
public static final java.lang.String TAGGEDPROFILESEQ_COD
```

Ruta donde se crea el fichero taggedProfileSeq_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

TAGGEDPROFILESEQ_DECO

```
public static final java.lang.String TAGGEDPROFILESEQ_DECO
```

Ruta donde se crea el fichero taggedProfileSeq_deco.xml

See Also:
Constant Field Values

TRAITS_KNOWN_COD

```
public static final java.lang.String TRAITS_KNOWN_COD
```

Ruta donde se crea el fichero traits_known_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

TRAITS_KNOWN_DECO

```
public static final java.lang.String TRAITS_KNOWN_DECO
```

Ruta donde se crea el fichero traits_known_deco.xml

See Also:
Constant Field Values

Constructor Detail

Constantes2

```
public Constantes2()
```

Server.WebServiceClasses
Class IdentifyPersonWS
java.lang.Object
  Server.WebServiceClasses.IdentifyPersonWS

All Implemented Interfaces:
IdentifyPerson

```
public class IdentifyPersonWSextends java.lang.Objectimplements
IdentifyPerson
```

Clase que implementa la interfaz IdentifyPerson del Servicio Web.

Author:
Pilar Mª Juan Montilla

Constructor Summary

IdentifyPersonWS()	

Method Summary

org.apache.axiom.om.OMElement	**find_candidates**(org.apache.axiom.om.OMElement peticion)

	Método que recibe todos los parámetros necesarios para invocar al método find_candidates de la clase IdentifyPersonXML.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

IdentifyPersonWS

```
public IdentifyPersonWS()
```

Method Detail

find_candidates

```
public org.apache.axiom.om.OMElement
find_candidates(org.apache.axiom.om.OMElement peticion)
```

Description copied from interface: `IdentifyPerson`
Método que recibe todos los parámetros necesarios para invocar al método find_candidates de la clase IdentifyPersonXML.

Specified by:
`find_candidates` in interface `IdentifyPerson`

Parameters:
`peticion` - OMElement con todos los parámetros de entrada.

Returns:
OMElement de respuesta con los documentos de salida returned_sequence_cod.xml y returned_iterator_cod.xml.

Server.WebServiceClasses
Class ProfileAccessWS
java.lang.Object
  Server.WebServiceClasses.ProfileAccessWS

All Implemented Interfaces:
ProfileAccess

```
public class ProfileAccessWSextends java.lang.Objectimplements
ProfileAccess
```

Clase que implementa la interfaz ProfileAccess del Servicio Web.

Author:
Pilar Mª Juan Montilla

Constructor Summary

ProfileAccessWS()	

Method Summary

org.apache.axiom.om.OMElement	**get_profile_list**(org.apache.axiom.om.OMElement entrada) Método que recibe todos los parámetros necesarios para invocar al método get_profile_list de la clase ProfileAccessXML.
org.apache.axiom.om.OMElement	**get_profile**(org.apache.axiom.om.OMElement entrada) Método que recibe todos los parámetros necesarios para invocar al método get_profile de la clase ProfileAccessXML.
org.apache.axiom.om.OMElement	**get_traits_known**(org.apache.axiom.om.OMElement id) Método que recibe todos los parámetros necesarios para invocar al método get_traits_known de la clase ProfileAccessXML.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ProfileAccessWS

```
public ProfileAccessWS()
```

Method Detail

get_profile

```
public org.apache.axiom.om.OMElement
get_profile(org.apache.axiom.om.OMElement entrada)
```

Description copied from interface: ProfileAccess
Método que recibe todos los parámetros necesarios para invocar al método get_profile de la clase ProfileAccessXML.

Specified by:
get_profile in interface ProfileAccess
Parameters:
entrada - OMElement con todos los parámetros de entrada: uno identificador de persona y el contenido del documento de entrada traits_requested_cod.xml
Returns:
OMElement de respuesta con el contenido del fichero de salida profile_cod.xml

---

get_profile_list

```
public org.apache.axiom.om.OMElement
get_profile_list(org.apache.axiom.om.OMElement entrada)
```

Description copied from interface: ProfileAccess
Método que recibe todos los parámetros necesarios para invocar al método get_profile_list de la clase ProfileAccessXML.

Specified by:
get_profile_list in interface ProfileAccess
Parameters:
entrada - OMElement con todos los parámetros de entrada: uno o varios identificadores de persona y el contenido del documento de entrada traits_requested.xml
Returns:
OMElement de respuesta con el contenido del fichero de salida taggedProfileSeq_cod.xml

---

get_traits_known

```
public org.apache.axiom.om.OMElement
get_traits_known(org.apache.axiom.om.OMElement id)
```

Description copied from interface: ProfileAccess

Método que recibe todos los parámetros necesarios para invocar al método get_traits_known de la clase ProfileAccessXML.

Specified by:

`get_traits_known` in interface `ProfileAccess`

Parameters:

`id` - OMElement con un identificador de persona.

Returns:

OMElement de respuesta con el contenido del fichero de salida traits_known_cod.xml

---

13.2 Cliente

Paquetes de la Aplicación

Packages	
Client	Este paquete contiene el servlet y el resto de las clases java que éste utiliza para recopilar la información necesaria para preparar la petición e invocar al servidor.
Codification	Paquete donde se encuentra la clase Canonizador.java (también usada en el servidor) encargada de codificar los documentos XML para que a priori no contengan caracteres no válidos como pudieran ser las 'ñ' o las tildes.
PID.DataTypes	Este paquete contiene las clases que encapsulan a los tipos de datos definidos en el PID en el apartado de Data Types.

Figura 13.3: Paquetes del Cliente.

Paquete Client

Class Summary	
Auxiliar	Clase donde se implementan los métodos crea_xml y get_OMELement.
Client	Esta clase es el servlet de la aplicación web cliente.
Constantes	Clase donde se definen las constantes con las rutas de los documentos XML que se crean en el cliente.
Create_XML	Clase que se encarga de crear los ficheros profile_selector.xml y traits_requested.xml a partir de la información recuperada de los formularios.
IdentifyPersonClient	Clase que se encarga de invocar al método find_candidates de la interfaz IdentifyPerson del Servicio Web.
ProfileAccessClient	Clase que invoca a los métodos get_traits_known, get_profile y get_profile_list de la interfaz ProfileAccess del Servicio Web.

Figura 13.4: Paquete Client.

Client
Class Auxiliar
java.lang.Object
  └**Client.Auxiliar**

---

public class **Auxiliar**

extends java.lang.Object

Clase donde se implementan los métodos crea_xml y get_OMELement.

Author:
Pilar Mª Juan Montilla

Constructor Summary

Auxiliar()	

Method Summary

static void	**crea_xml**(java.lang.String contenido_xml, java.lang.String ruta_fichero) Método que crea un fichero XML a partir de su contenido y su ruta absoluta.
static org.apache.axiom.om.OMElement	**get_OMElement**(java.lang.String ruta_fichero_xml) Método que obtiene el OMElement correspondiente a un fichero XML.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Auxiliar

public **Auxiliar**()

Method Detail

crea_xml

```
public static void crea_xml(java.lang.String contenido_xml,
                            java.lang.String ruta_fichero)
```

Método que crea un fichero XML a partir de su contenido y su ruta absoluta.

Parameters:

`contenido_xml` - String con el contenido del fichero XML a crear.
`ruta_fichero` - String con la ruta del fichero XML a crear.

---

get_OMElement

```
public static org.apache.axiom.om.OMElement get_OMElement(java.lang.String
ruta_fichero_xml)
```

Método que obtiene el OMElement correspondiente a un fichero XML.

Parameters:

`ruta_fichero_xml` - String que contiene la ruta del fichero XML en cuestión.

Returns:

El OMELement de salida

Client
Class Client

```
java.lang.Object
  └javax.servlet.GenericServlet
      └javax.servlet.http.HttpServlet
          └Client.Client
```

All Implemented Interfaces:

java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

```
public class Clientextends javax.servlet.http.HttpServlet
```

Esta clase es el servlet de la aplicación web cliente.

Author:

Pilar Mª Juan Montilla

See Also:
Serialized Form

Constructor Summary

Client()	

Method Summary

void	**doGet**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
void	**doPost**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
java.lang.String[]	**obtiene_ids**(javax.servlet.http.HttpServletRequest request) Método que recibe la petición HTTP que hace el usuario remoto a través del formulario y recupera todos los ids que se han indicado en una tabla de Strings.

Methods inherited from class javax.servlet.http.HttpServlet
service

Methods inherited from class javax.servlet.GenericServlet
destroy, getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, init, log, log

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Client

```
public Client()
```

Method Detail

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

Overrides:

`doGet` in class `javax.servlet.http.HttpServlet`

Throws:

`javax.servlet.ServletException`
`java.io.IOException`

---

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,
                   javax.servlet.http.HttpServletResponse response)
```

Overrides:

`doPost` in class `javax.servlet.http.HttpServlet`

---

obtiene_ids

```
public java.lang.String[] obtiene_ids(javax.servlet.http.HttpServletRequest
request)
```

Método que recibe la petición HTTP que hace el usuario remoto a través del formulario y recupera todos los ids que se han indicado en una tabla de Strings.

Parameters:

`request` - Es la petición HTTP.

Returns:

Tabla de Strings con todos los ids que se han especificado en la petición. en el formulario.

Client
Class Constantes
java.lang.Object
  └**Client.Constantes**

---

public class **Constantes**extends java.lang.Object

Clase donde se definen las constantes con las rutas de los documentos XML que se crean en el cliente.

Author:
Pilar Mª Juan Montilla

Field Summary	
static java.lang.String	**PIDS** Ruta del mapa PIDs.xml, necesario para la transformación Java-XML mediante el uso de Castor
static java.lang.String	**PROFILE** Ruta donde se crea el fichero profile.xml
static java.lang.String	**PROFILE_COD** Ruta donde se crea el fichero profile_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
static java.lang.String	**PROFILE_SELECTOR** Ruta donde se crea el fichero profile_selector.xml
static java.lang.String	**PROFILE_SELECTOR_COD** Ruta donde se crea el fichero profile_selector_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
static java.lang.String	**RETURNED_ITERATOR** Ruta donde se crea el fichero returned_iterator.xml
static java.lang.String	**RETURNED_ITERATOR_COD** Ruta donde se crea el fichero returned_iterator_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
static java.lang.String	**RETURNED_SEQUENCE** Ruta donde se crea el fichero returned_sequence.xml
static java.lang.String	**RETURNED_SEQUENCE_COD** Ruta donde se crea el fichero returned_sequence_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
static java.lang.String	**TAGGEDPROFILESEQ**

	Ruta donde se crea el fichero taggedProfileSeq.xml
`static java.lang.String`	**TAGGEDPROFILESEQ_COD** Ruta donde se crea el fichero taggedProfileSeq_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
`static java.lang.String`	**TRAITS_KNOWN** Ruta donde se crea el fichero traits_known.xml
`static java.lang.String`	**TRAITS_KNOWN_COD** Ruta donde se crea el fichero traits_known_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')
`static java.lang.String`	**TRAITS_REQUESTED** Ruta donde se crea el fichero traits_requested.xml
`static java.lang.String`	**TRAITS_REQUESTED_COD** Ruta donde se crea el fichero traits_requested_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

Constructor Summary

Constantes()	

Method Summary

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

PROFILE_SELECTOR

```
public static final java.lang.String PROFILE_SELECTOR
```

Ruta donde se crea el fichero profile_selector.xml

See Also:
Constant Field Values

PROFILE_SELECTOR_COD

```
public static final java.lang.String PROFILE_SELECTOR_COD
```

Ruta donde se crea el fichero profile_selector_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

TRAITS_REQUESTED

```
public static final java.lang.String TRAITS_REQUESTED
```

Ruta donde se crea el fichero traits_requested.xml

See Also:
Constant Field Values

TRAITS_REQUESTED_COD

```
public static final java.lang.String TRAITS_REQUESTED_COD
```

Ruta donde se crea el fichero traits_requested_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

RETURNED_SEQUENCE

```
public static final java.lang.String RETURNED_SEQUENCE
```

Ruta donde se crea el fichero returned_sequence.xml

See Also:
Constant Field Values

RETURNED_SEQUENCE_COD

```
public static final java.lang.String RETURNED_SEQUENCE_COD
```

Ruta donde se crea el fichero returned_sequence_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

RETURNED_ITERATOR

```
public static final java.lang.String RETURNED_ITERATOR
```

Ruta donde se crea el fichero returned_iterator.xml

See Also:
Constant Field Values

RETURNED_ITERATOR_COD

```
public static final java.lang.String RETURNED_ITERATOR_COD
```

Ruta donde se crea el fichero returned_iterator_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

PROFILE

```
public static final java.lang.String PROFILE
```

Ruta donde se crea el fichero profile.xml

See Also:
Constant Field Values

PROFILE_COD

```
public static final java.lang.String PROFILE_COD
```

Ruta donde se crea el fichero profile_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

TRAITS_KNOWN

```
public static final java.lang.String TRAITS_KNOWN
```

Ruta donde se crea el fichero traits_known.xml

See Also:
Constant Field Values

TRAITS_KNOWN_COD

```
public static final java.lang.String TRAITS_KNOWN_COD
```

Ruta donde se crea el fichero traits_known_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

TAGGEDPROFILESEQ

```
public static final java.lang.String TAGGEDPROFILESEQ
```

Ruta donde se crea el fichero taggedProfileSeq.xml

See Also:
Constant Field Values

TAGGEDPROFILESEQ_COD

```
public static final java.lang.String TAGGEDPROFILESEQ_COD
```

Ruta donde se crea el fichero taggedProfileSeq_cod.xml ("cod" de codificado, es decir, sin acentos ni caracteres 'ñ')

See Also:
Constant Field Values

PIDS

```
public static final java.lang.String PIDS
```

Ruta del mapa PIDs.xml, necesario para la transformación Java-XML mediante el uso de Castor

See Also:
Constant Field Values

Constructor Detail

Constantes

```
public Constantes()
```

Client
Class IdentifyPersonClient

```
java.lang.Object
  └Client.IdentifyPersonClient
```

```
public class IdentifyPersonClientextends java.lang.Object
```

Clase que se encarga de invocar al método find_candidates de la interfaz IdentifyPerson del Servicio Web.

Author:
Pilar Mª Juan Montilla

Constructor Summary

IdentifyPersonClient()	

Method Summary

org.apache.axiom.om.OMElement	**crea_OMElement_PI**(java.lang.String contenido_profile_selector, IdStateSeq states_of_interest_real, float confidence_threshold_real, long sequence_max_real, long

	iterator_max_real, java.lang.String contenido_traits_requested) Método que recibe todos los parámetros que se necesitan para invocar al método fin_candidates del Servicio Web y se encarga de preparar el OMElement de petición.
java.lang.String	**find_candidates_cli**(java.lang.Strin ruta_profile_selector, IdStateSeq_ states_of_interest, float confidence_threshold, long sequence_max, long iterator_max, java.lang.String ruta_traits_requested) Método que recibe todos los datos recopilados de los formularios de entrada para invocar al método find_candidates de la interfaz IdentifyPerson del Servicio Web.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

IdentifyPersonClient

```
public IdentifyPersonClient()
```

Method Detail

find_candidates_cli

```
public java.lang.String find_candidates_cli(java.lang.String
ruta_profile_selector, IdStateSeq_states_of_interest,float
confidence_threshold, long sequence_max, long iterator_max,
java.lang.String ruta_traits_requested)
```

Método que recibe todos los datos recopilados de los formularios de entrada para invocar al método find_candidates de la interfaz IdentifyPerson del Servicio Web.

Parameters:

`ruta_profile_selector` - Ruta del fichero de entrada profile_selector.xml.
`states_of_interest` - Estados admitidos para el identificador de la persona (no implementado).
`confidence_threshold` - Confianza minima requerida en el candidato

devuelto.
`sequence_max` - Número máximo de candidatos devueltos en returned_secuence.xml
`iterator_max` - Número maximo de candidatos devueltos en returned_iterator.xml
`ruta_traits_requested` - Ruta del fichero de entrada traits_requested.xml.

Returns:

Ruta relativa del fichero de salida.

---

crea_OMElement_PI

```
public org.apache.axiom.om.OMElement crea_OMElement_PI(java.lang.String
contenido_profile_selector, IdStateSeq states_of_interest_real, float
confidence_threshold_real, long sequence_max_real, long iterator_max_real,
java.lang.String contenido_traits_requested)
```

Método que recibe todos los parámetros que se necesitan para invocar al método fin_candidates del Servicio Web y se encarga de preparar el OMElement de petición.

Parameters:

`contenido_profile_selector` - String con el contenido del fichero de entrada profile_selector_cod.xml
`states_of_interest_real` -
`confidence_threshold_real` -
`sequence_max_real` - Long con el número máximo de candidatos devueltos en returned_secuence.xml
`iterator_max_real` - Long con el número máximo de candidatos devueltos en returned_iterator.xml
`contenido_traits_requested` - String con el contenido del fichero de entrada traits_requested_cod.xml

Returns:

OMElement con la petición para invocar al método find_candidates del Servicio Web.

Client
Class ProfileAccessClient
java.lang.Object
  └**Client.ProfileAccessClient**

---

public class **ProfileAccessClient**extends java.lang.Object

Clase que invoca a los métodos get_traits_known, get_profile y get_profile_list de la interfaz ProfileAccess del Servicio Web.

Author:
Pilar Mª Juan Montilla

Constructor Summary

ProfileAccessClient()	

Method Summary

java.lang.String	**get_profile_cli**(java.lang.String id, java.lang.String ruta_traits_requested) Método que recibe todos los datos necesarios para invocar al método get_profile de la interfaz ProfileAccess del Servicio Web.
java.lang.String	**get_profile_list_cli**(java.lang.String[] ids, java.lang.String ruta_traits_requested) Método que recibe todos los datos necesarios para invocar al método get_profile_list de la interfaz ProfileAccess del Servicio Web.
java.lang.String	**get_traits_known_cli**(java.lang.String id) Método que recibe todos los datos necesarios para invocar al método get_traits_known de la interfaz ProfileAccess del Servicio Web.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ProfileAccessClient

```
public ProfileAccessClient()
```

Method Detail

get_traits_known_cli

```
public java.lang.String get_traits_known_cli(java.lang.String id)
```

Método que recibe todos los datos necesarios para invocar al método get_traits_known de la interfaz ProfileAccess del Servicio Web.

Parameters:

`id` - String que contiene el identificador de una persona.

Returns:

Ruta relativa del fichero de salida traits_known.xml.

---

get_profile_cli

```
public java.lang.String get_profile_cli(java.lang.String id,
java.lang.String ruta_traits_requested)
```

Método que recibe todos los datos necesarios para invocar al método get_profile de la interfaz ProfileAccess del Servicio Web.

Parameters:

`id` - String que contiene el identificador de una persona.

`ruta_traits_requested` - Ruta del fichero de entrada traits_requested.xml

Returns:

Ruta relativa del fichero de salida profile.xml

---

get_profile_list_cli

```
public java.lang.String get_profile_list_cli(java.lang.String[] ids,
java.lang.String ruta_traits_requested)
```

Método que recibe todos los datos necesarios para invocar al método get_profile_list de la interfaz ProfileAccess del Servicio Web.

Parameters:

`ids` - Tabla de Strings con el identificador de una persona o varias personas.

`ruta_traits_requested` - Ruta del fichero de entrada traits_requested.xml

Returns:

Ruta relativa del fichero de salida taggedProfileSeq.xml

PARTE VIII: CÓDIGO DE LA APLICACIÓN

14 Código de la Aplicación

14.1 Servidor

IdentifyPerson.java

```
package Server.WebServiceClasses;

import org.apache.axiom.om.OMElement;

/**
 * Clase en la que se define la interfaz IdentifyPerson del Servicio Web.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public interface IdentifyPerson {


      /**
       * Método que recibe todos los parámetros necesarios para invocar al
método find_candidates de la clase IdentifyPersonXML.
       *
       * @param peticion OMElement con todos los parámetros de entrada.
       * @return OMElement de respuesta con los documentos de salida
returned_sequence_cod.xml y returned_iterator_cod.xml.
       */
      public OMElement find_candidates ( OMElement peticion );

}


//Según la especificación PIDS:
//void find_candidates (TraitSelectorSeq profile_selector, IdStateSeq
//states_of_interest, float confidence_threshold, long sequence_max, long
//iterator_max, SpecifiedTraits traits_requested,
//CandidateSeq returned_sequence, CandidateIterator returned_iterator);
```

IdentifyPersonWS.java

```
package Server.WebServiceClasses;

import Server.WebServiceClasses.Auxiliar;
import java.io.File;

import javax.xml.namespace.QName;
import javax.xml.stream.XMLStreamException;
```

```
import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;

import Server.Local.PID.DataBase.DB_Manager;
import Server.PID.DataTypes.*;
import Server.PID.Exception.DuplicateTraits;
import Server.PID.Exception.InvalidWeight;
import Server.PID.Exception.UnknownTraits;
import Server.PID.Implementation.IdentifyPersonXML;

/**
 * Clase que implementa la interfaz IdentifyPerson del Servicio Web.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class IdentifyPersonWS implements IdentifyPerson{


      public OMElement find_candidates (OMElement peticion){


            //SE OBTIENE EL CONTENIDO Y CREAMOS LOS FICHEROS XML
            //profile_selector_cod.xml y traits_requested_cod.xml DE
            //ENTRADA A PARTIR DE LOS OMELEMENT QUE SE RECIBEN DEL CLIENTE,
            //ASÍ COMO LOS DEMÁS PARÁMETROS DE ENTRADA DEL MÉTODO

            OMElement OM_profile_selector= peticion.getFirstElement();
            String contenido_profile_selector_cod =
                  OM_profile_selector.getText();
            System.out.println("El contenido del fichero
profile_selector_cod.xml es el siguiente:\n"
+contenido_profile_selector_cod);
            //CREAMOS profile_selector_cod.xml
            Auxiliar.crea_xml(contenido_profile_selector_cod,
                  Constantes2.PROFILE_SELECTOR_COD);


            OMElement OM_states_of_interest=
peticion.getFirstChildWithName(new QName("http://service/xsd",
"states_of_interest"));
            String ST_states_of_interest = OM_states_of_interest.getText();
            System.out.println("states_of_interest: "+
                  ST_states_of_interest);
            IdStateSeq states_of_interest=null;
            if (ST_states_of_interest.equals("null")){
                  states_of_interest=null; //obvio
                  //como no está implementado tendrá que valer null
            }
             else
                  System.out.println("Error al especificar el estado de
interés");

            OMElement OM_confidence_threshold=
```

```
                  (OMElement)OM_states_of_interest.getNextOMSibling();
            String ST_confidence_threshold =
                  OM_confidence_threshold.getText();
            System.out.println("confidence_threshold: "+
ST_confidence_threshold);
            float confidence_threshold=new
                  Float(ST_confidence_threshold).floatValue();

            OMElement OM_sequence_max=
                  (OMElement)OM_confidence_threshold.getNextOMSibling();
            String ST_sequence_max = OM_sequence_max.getText();
            System.out.println("sequence_max: "+ST_sequence_max);
            long sequence_max=new Long(ST_sequence_max).longValue();

            OMElement OM_iterator_max=
                  (OMElement)OM_sequence_max.getNextOMSibling();
            String ST_iterator_max = OM_iterator_max.getText();
            System.out.println("iterator_max: "+ST_iterator_max);
            long iterator_max=new Long(ST_iterator_max).longValue();

            OMElement OM_traits_requested=
            (OMElement)OM_iterator_max.getNextOMSibling();
            String contenido_traits_requested_cod =
                  OM_traits_requested.getText();
            System.out.println("El contenido del fichero
traits_requested_cod.xml es el siguiente:\n"+
contenido_traits_requested_cod);

            //CREAMOS traits_requested_cod.xml
            Auxiliar.crea_xml(contenido_traits_requested_cod,
                  Constantes2.TRAITS_REQUESTED_COD);


            //SE ABREN LOS DOS FICHEROS XML DE ENTRADA
            File file_profile_selector_cod= new
                  File(Constantes2.PROFILE_SELECTOR_COD);
            File file_traits_requested_cod= new File
                  (Constantes2.TRAITS_REQUESTED_COD);

            //SE ABREN LOS DOS FICHEROS XML DE SALIDA
            File file_returned_sequence_cod =new
                  File(Constantes2.RETURNED_SEQUENCE_COD);
            File file_returned_iterator_cod =new
                  File(Constantes2.RETURNED_ITERATOR_COD);

            //NOS CONECTAMOS A LA BASE DE DATOS (Suponiendo ya creada la
            //base de datos)
            DB_Manager DBM=new DB_Manager();
            //Cargando el controlador
            DBM.loadDriver();
            //Haciendo la conexión
            DBM.doConnection();


            //Se crea el objeto que implementa la interfaz IdentifyPerson
            IdentifyPersonXML IdPersonXML=new IdentifyPersonXML(DBM);

             System.out.println("Llamamos al método find_candidates");
```

```
             try {


IdPersonXML.find_candidates(file_profile_selector_cod,states_of_interest,co
nfidence_threshold,sequence_max, iterator_max
, file_traits_requested_cod, file_returned_sequence_cod,
file_returned_iterator_cod);

            } catch (UnknownTraits e) {

                  System.out.println("UnknownTraits"+e);
                  e.printStackTrace();
                  System.exit(1);

            } catch (InvalidWeight e) {

                  System.out.println("InvaliWeight"+e);
                  e.printStackTrace();
                  System.exit(1);

            } catch (DuplicateTraits e) {

                  System.out.println("DuplicateTraits"+e);
                  e.printStackTrace();
                  System.exit(1);
            }

            //CREAMOS LOS OBJETOS OMELEMENT DE SALIDA A PARTIR DE LOS
            //FICHEROS returned_sequence_cod.xml y
            //returned_iterator_cod.xml
            //OBTENIDOS AL EJECUTAR EL MÉTODO find_candidates

            OMElement returned_sequence =
            Auxiliar.get_OMElement(Constantes2.RETURNED_SEQUENCE_COD);

            OMElement returned_iterator =
            Auxiliar.get_OMElement(Constantes2.RETURNED_ITERATOR_COD);

            String contenido_returned_sequence_cod=null;
            String contenido_returned_iterator_cod=null;

            try {

                  contenido_returned_sequence_cod =
                        returned_sequence.toStringWithConsume();
                  contenido_returned_iterator_cod =
                        returned_iterator.toStringWithConsume();

                  System.out.println("El contenido del fichero
returned_sequence_cod.xml es el siguiente:\n"+
contenido_returned_sequence_cod);
                  System.out.println("El contenido del fichero
returned_iterator_cod.xml es el siguiente:\n"+
contenido_returned_iterator_cod);

            } catch (XMLStreamException e) {
```

```
            System.out.println("Se ha producido el siguiente error en la
clase del servicio:\n");
             e.printStackTrace();
             System.exit(1);
            }

            //SE CONSTRUYE EL ELEMENTO OMELEMENT DE SALIDA
            OMFactory fac = OMAbstractFactory.getOMFactory();
            OMNamespace omNs =
            fac.createOMNamespace("http://service/xsd","IdentifyPerson");
            OMElement salida = fac.createOMElement("respuesta", omNs);

            OMElement OM_returned_sequence =
            fac.createOMElement("returned_sequence", omNs);

      OM_returned_sequence.addChild(fac.createOMText(OM_returned_sequence,
contenido_returned_sequence_cod));
            salida.addChild(OM_returned_sequence);

            OMElement OM_returned_iterator =
                  fac.createOMElement("returned_iterator", omNs);

      OM_returned_iterator.addChild(fac.createOMText(OM_returned_iterator,
contenido_returned_iterator_cod));
            salida.addChild(OM_returned_iterator);

            //Se cierra la conexión con la base de datos
            DBM.closeConnection();


            return salida;

      }


}
```

ProfileAccess.java

```
package Server.WebServiceClasses;

import org.apache.axiom.om.OMElement;

/**
 * Clase en la que se define la interfaz ProfileAccess del Servicio Web.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public interface ProfileAccess {

      /**
       * Método que recibe todos los parámetros necesarios para invocar al
```

```
método get_profile de la clase ProfileAccessXML.
       *
       * @param entrada OMElement con todos los parámetros de entrada: uno
identificador de persona y el contenido del documento de entrada
traits_requested_cod.xml
       * @return OMElement de respuesta con el contenido del fichero de
salida profile_cod.xml
       */
      public OMElement get_profile ( OMElement entrada );

      // Método de la especificación:
      //Profile get_profile ( Person id, SpecifiedTraits traits_requested )


      /**
       * Método que recibe todos los parámetros necesarios para invocar al
método get_profile_list de la clase ProfileAccessXML.
       *
       * @param entrada  OMElement con todos los parámetros de entrada: uno
o varios identificadores de persona y el contenido del documento de entrada
traits_requested.xml
       * @return OMElement de respuesta con el contenido del fichero de
salida taggedProfileSeq_cod.xml
       */
      public OMElement get_profile_list ( OMElement entrada );

      // Método de la especificación:
      //TaggedProfileSeq get_profile( PersonIdSeq ids, SpecifiedTraits
      //traits_requested )


      /**
       * Método que recibe todos los parámetros necesarios para invocar al
método get_traits_known de la clase ProfileAccessXML.
       *
       * @param id  OMElement con un identificador de persona.
       * @return OMElement de respuesta con el contenido del fichero de
salida traits_known_cod.xml
       */
      public OMElement get_traits_known ( OMElement id );

      // Método de la especificación:
      //TraitNameSeq get_traits_known ( PersonId id )

}
```

ProfileAccessWS.java

```
package Server.WebServiceClasses;

import Server.WebServiceClasses.Auxiliar;
```

```
import java.io.File;

import javax.xml.stream.XMLStreamException;

import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;

import Server.Local.PID.DataBase.DB_Manager;
import Server.PID.Exception.DuplicateIds;
import Server.PID.Exception.DuplicateTraits;
import Server.PID.Exception.InvalidId;
import Server.PID.Exception.InvalidIds;
import Server.PID.Exception.UnknownTraits;
import Server.PID.Implementation.ProfileAccessXML;

/**
 * Clase que implementa la interfaz ProfileAccess del Servicio Web.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class ProfileAccessWS implements ProfileAccess{


      public OMElement get_profile(OMElement entrada) {


            //NOS CONECTAMOS A LA BASE DE DATOS (Suponiendo ya creada la
            //base de datos)
            DB_Manager DBM=new DB_Manager();
            //Cargando el controlador
            DBM.loadDriver();
            //Haciendo la conexión
            DBM.doConnection();

            //RECUPERAMOS LOS PARÁMETROS DE ENTRADA
            OMElement OM_id= entrada.getFirstElement();
            String id= OM_id.getText();
            System.out.println("El id solicitado es: "+id);

            OMElement traits_requested=
                  (OMElement) OM_id.getNextOMSibling();
            String contenido_traits_requested_cod =
                  traits_requested.getText();
            System.out.println("El contenido del fichero
traits_requested_cod es el siguiente:\n"+contenido_traits_requested_cod);

            //CREAMOS  Y ABRRIMOS EL FICHERO traits_requested_cod.xml
            Auxiliar.crea_xml(contenido_traits_requested_cod,
                  Constantes2.TRAITS_REQUESTED_COD);
            File file_traits_requested_cod=
                  new File (Constantes2.TRAITS_REQUESTED_COD);

            //CREAMOS EL FICHERO DE SALIDA profile_cod.xml
            File file_profile=null;
```

```
            //Se crea el objeto que implementa la interfaz ProfileAccess
            ProfileAccessXML ProAccXML=new ProfileAccessXML(DBM);

            try {


                  file_profile=ProAccXML.get_profile(id,
                        file_traits_requested_cod);

            } catch (InvalidId e) {

                  System.out.println("InvalidId:\n");
                  e.printStackTrace();
                  System.exit(1);

            } catch (UnknownTraits e) {

                  System.out.println("UnknownTRaits:\n");
                  e.printStackTrace();
                  System.exit(1);

            }

            OMElement salida=null;

            if (file_profile!=null)
            {
                  OMElement profile =
                        Auxiliar.get_OMElement(Constantes2.PROFILE_COD);

                  String contenido_profile_cod=null;

                  try {

                        contenido_profile_cod=
                              profile.toStringWithConsume();
                        System.out.println("El contenido del fichero
profile_cod.xml es el siguiente:\n"+contenido_profile_cod);

                  } catch (XMLStreamException e) {

                        e.printStackTrace();
                        System.exit(1);
                  }

                  //SE CONSTRUYE EL ELEMENTO OMELEMENT DE SALIDA
                  OMFactory fac = OMAbstractFactory.getOMFactory();
                  OMNamespace omNs =
            fac.createOMNamespace("http://service/xsd","ProfileAccess");
                  salida = fac.createOMElement("respuesta", omNs);

                  OMElement OM_profile=
                         fac.createOMElement("profile",omNs);
                  OM_profile.addChild(fac.createOMText(OM_profile,
                        contenido_profile_cod));
                  salida.addChild(OM_profile);
```

```
            }
            else
                  System.out.println("El servicio ProfileAccess no devuelve
nada");

            //Se cierra la conexion a la base de datos
            DBM.closeConnection();

            return salida;
      }

      public OMElement get_profile_list (OMElement entrada ){

            //NOS CONECTAMOS A LA BASE DE DATOS (Suponiendo ya creada la
            //base de datos)
            DB_Manager DBM=new DB_Manager();
            //Cargando el controlador
            DBM.loadDriver();
            //Haciendo la conexión
            DBM.doConnection();


            int indice=0;

            //RECUPERAMOS LOS PARÁMETROS DE ENTRADA
            OMElement OM_num_ids= entrada.getFirstElement();
            String ST_num_ids= OM_num_ids.getText();
            System.out.println("El número de ids solicitados es:
"+ST_num_ids);
            int num_ids= new Integer(ST_num_ids).intValue();

            String ids[]= new String[num_ids];

            OMElement OM_id=null;
            OMElement anterior=null;
            anterior= OM_num_ids;

            while (num_ids!=0){

                  OM_id= (OMElement) anterior.getNextOMSibling();
                  String ST_id= OM_id.getText();
                  ids[indice]=new String(ST_id);
                  System.out.println("id["+indice+"] vale: "+ST_id);
                  anterior=OM_id;
                  indice=indice+1;
                  num_ids=num_ids-1;
            }

            OMElement traits_requested=
                  (OMElement) anterior.getNextOMSibling();
            String contenido_traits_requested_cod =
                  traits_requested.getText();
            System.out.println("El contenido del fichero
traits_requested_cod es el siguiente:\n"+contenido_traits_requested_cod);

            //SE CREA  Y SE ABRE EL FICHERO traits_requested_cod.xml
            Auxiliar.crea_xml(contenido_traits_requested_cod,
                  Constantes2.TRAITS_REQUESTED_COD);
```

```
            File file_traits_requested_cod=
                  new File (Constantes2.TRAITS_REQUESTED_COD);

            //CREAMOS EL FICHERO DE SALIDA taggedProfileSeq_cod.xml
            File file_taggedProfileSeq_cod=null;

            //Creamos el objeto que implementa la interfaz ProfileAccess
            ProfileAccessXML ProAccXML=new ProfileAccessXML(DBM);

            try {

                  file_taggedProfileSeq_cod=ProAccXML.get_profile_list(ids,
                        file_traits_requested_cod);

            } catch (InvalidIds e) {
                  System.out.println("InvalidIds:\n");
                  e.printStackTrace();
                  System.exit(1);

            } catch (DuplicateIds e) {
                  System.out.println("DuplicateIds:\n");
                  e.printStackTrace();
                  System.exit(1);

            } catch (UnknownTraits e) {
                  System.out.println("UnknownTraits:\n");
                  e.printStackTrace();
                  System.exit(1);

            } catch (DuplicateTraits e) {
                  System.out.println("DuplicateTraits:\n");
                  e.printStackTrace();
                  System.exit(1);
            }

            OMElement salida=null;

            if (file_taggedProfileSeq_cod!=null)
            {
                  OMElement taggedProfileSeq =
      Auxiliar.get_OMElement(Constantes2.TAGGEDPROFILESEQ_COD);

                  String contenido_taggedProfileSeq_cod=null;

                  try {

                        contenido_taggedProfileSeq_cod=
                              taggedProfileSeq.toStringWithConsume();
                        System.out.println("El contenido del fichero
taggedProfileSeq_cod.xml es el siguiente:\n"+
contenido_taggedProfileSeq_cod);

                  } catch (XMLStreamException e) {

                        e.printStackTrace();
                        System.exit(1);
                  }
```

```
                  //SE CONSTRUYE EL ELEMENTO OMELEMENT DE SALIDA
                  OMFactory fac = OMAbstractFactory.getOMFactory();
                  OMNamespace omNs =
            fac.createOMNamespace("http://service/xsd","ProfileAccess");
                  salida = fac.createOMElement("respuesta", omNs);

                  OMElement OM_taggedProfileSeq=
                        fac.createOMElement("taggedProfileSeq", omNs);

      OM_taggedProfileSeq.addChild(fac.createOMText(OM_taggedProfileSeq,
contenido_taggedProfileSeq_cod));
                  salida.addChild(OM_taggedProfileSeq);

            }
            else
                  System.out.println("El servicio ProfileAccess no devuelve
nada");

            // Se cierra la conexion a la base de datos
            DBM.closeConnection();

            return salida;


      }


      public OMElement get_traits_known(OMElement id) {

            //NOS CONECTAMOS A LA BASE DE DATOS (Suponiendo ya creada la
            //base de datos)

            DB_Manager DBM=new DB_Manager();
            //Cargando el controlador
            DBM.loadDriver();
            //Haciendo la conexión
            DBM.doConnection();

            //RECUPERAMOS EL PARÁMETRO DE ENTRADA: ID
            OMElement OM_id= id.getFirstElement();
            String ST_id= OM_id.getText();
            System.out.println("El id solicitado es: "+ST_id);

            //CREAMOS EL FICHERO DE SALIDA traits_known_cod.xml
            File file_traits_known_cod=null;

            //Creamos el objeto que implementa la interfaz ProfileAccess
            ProfileAccessXML ProAccXML=new ProfileAccessXML(DBM);

            try
            {
                  file_traits_known_cod=ProAccXML.get_traits_known(ST_id);

            }
            catch(InvalidId e){

                  System.out.println("InvalidId:\n"+e);
```

```
                e.printStackTrace();
                System.exit(1);
            }

            OMElement salida=null;

            if (file_traits_known_cod!=null){

            OMElement traits_known =
                Auxiliar.get_OMElement(Constantes2.TRAITS_KNOWN_COD);

            String contenido_traits_known_cod=null;

            try {

                contenido_traits_known_cod=
                      traits_known.toStringWithConsume();
                System.out.println("El contenido del fichero
traits_known_cod.xml es el siguiente:\n"+contenido_traits_known_cod);

            } catch (XMLStreamException e) {

                e.printStackTrace();
                System.exit(1);
            }

            //SE CONSTRUYE EL ELEMENTO OMELEMENT DE SALIDA
            OMFactory fac = OMAbstractFactory.getOMFactory();
            OMNamespace omNs =
            fac.createOMNamespace("http://service/xsd","ProfileAccess");
            salida = fac.createOMElement("respuesta", omNs);

            OMElement OM_traits_known=
                fac.createOMElement("traits_known", omNs);
            OM_traits_known.addChild(fac.createOMText(OM_traits_known,
                contenido_traits_known_cod));
            salida.addChild(OM_traits_known);

            }
            else
                System.out.println("El servicio ProfileAccess no devuelve
nada");

            //Se cierra la conexion a la base de datos
            DBM.closeConnection();

            return salida;

      }


}
```

Auxiliar.java

```
package Server.WebServiceClasses;

import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.impl.builder.StAXOMBuilder;

/**
 * Clase donde se implementan los métodos crea_xml y get_OMELement.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class Auxiliar {

      /**
       * Método que crea un fichero XML a partir de su contenido y su ruta
absoluta.
       *
       * @param contenido_xml String con el contenido del fichero XML a
crear.
       * @param ruta_fichero String con la ruta del fichero XML a crear.
       */
      public static void crea_xml (String contenido_xml, String
ruta_fichero){

            try {

                  FileWriter file_xml = new FileWriter (ruta_fichero);
                  BufferedWriter writer_xml= new BufferedWriter(file_xml);

                  writer_xml.write("<?xml version=\"1.0\" encoding=\"UTF-
8\"?>\n");
                  writer_xml.write(contenido_xml);
                  writer_xml.flush();

            } catch (IOException e) {

                  System.out.println("Se ha producido el siguiente error en
el método crea_xml:\n");
                  e.printStackTrace();
                  System.exit(1);

            }
      }
```

```
	/**
	 * Método que obtiene el OMElement correspondiente a un fichero XML.
	 *
	 * @param ruta_fichero_xml String que contiene la ruta del fichero
XML en cuestión.
	 * @return El OMELement de salida
	 */
	public static OMElement get_OMElement (String ruta_fichero_xml){

		OMElement salida=null;

		try {

			//create the parser
			XMLStreamReader parser =
XMLInputFactory.newInstance().createXMLStreamReader(new
FileInputStream(ruta_fichero_xml));
			StAXOMBuilder builder= new StAXOMBuilder(parser);
			//get the root element
			salida= builder.getDocumentElement();

		} catch (XMLStreamException e) {

			System.out.println("En el método get_OMElement se ha
producido el siguiente error:\n");
			e.printStackTrace();
			System.exit(1);

		} catch (FileNotFoundException e) {

			System.out.println("En el método get_OMElement se ha
producido el siquiente error:\n");
			e.printStackTrace();
			System.exit(1);
		}

		return salida;

	}


}
```

Constantes2.java

```
package Server.WebServiceClasses;

/**
 * Clase donde se definen las constantes con las rutas de los documentos
XML que se crean en el servidor.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class Constantes2 {

	/**
	 * Ruta donde se crea el fichero profile_selector_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
	 */
	public static final String
PROFILE_SELECTOR_COD="/root/workspace/PIDS-
WS/documentos/entradas/profile_selector_cod.xml";

	/**
	 * Ruta donde se crea el fichero profile_selector_deco.xml
	 */
	public static final String
PROFILE_SELECTOR_DECO="/root/workspace/PIDS-
WS/documentos/intermedios/profile_selector_deco.xml";

	/**
	 * Ruta donde se crea el fichero traits_requested_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
	 */
	public static final String
TRAITS_REQUESTED_COD="/root/workspace/PIDS-
WS/documentos/entradas/traits_requested_cod.xml";

	/**
	 * Ruta donde se crea el fichero traits_requested_deco.xml
	 */
	public static final String
TRAITS_REQUESTED_DECO="/root/workspace/PIDS-
WS/documentos/intermedios/traits_requested_deco.xml";

	/**
	 * Ruta donde se crea el fichero returned_sequence_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
	 */
	public static final String
RETURNED_SEQUENCE_COD="/root/workspace/PIDS-
WS/documentos/salidas/returned_sequence_cod.xml";

	/**
	 * Ruta donde se crea el fichero returned_sequence_deco.xml
	 */
	public static final String
```

```
RETURNED_SEQUENCE_DECO="/root/workspace/PIDS-
WS/documentos/intermedios/returned_sequence_deco.xml";

	/**
	 * Ruta donde se crea el fichero returned_iterator_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
	 */
	public static final String
RETURNED_ITERATOR_COD="/root/workspace/PIDS-
WS/documentos/salidas/returned_iterator_cod.xml";

	/**
	 * Ruta donde se crea el fichero returned_iterator_deco.xml
	 */
	public static final String
RETURNED_ITERATOR_DECO="/root/workspace/PIDS-
WS/documentos/intermedios/returned_iterator_deco.xml";

	/**
	 * Ruta del mapa PIDs.xml, necesario para la transformación Java-XML
mediante el uso de Castor
	 */
	public static final String PIDS="/root/workspace/PIDS-
WS/public_html/PIDs.xml";

	/**
	 * Ruta donde se crea el fichero profile_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
	 */
	public static final String PROFILE_COD="/root/workspace/PIDS-
WS/documentos/salidas/profile_cod.xml";

	/**
	 * Ruta donde se crea el fichero profile_deco.xml
	 */
	public static final String PROFILE_DECO="/root/workspace/PIDS-
WS/documentos/intermedios/profile_deco.xml";

	/**
	 * Ruta donde se crea el fichero taggedProfileSeq_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
	 */
	public static final String
TAGGEDPROFILESEQ_COD="/root/workspace/PIDS-
WS/documentos/salidas/taggedProfileSeq_cod.xml";

	/**
	 * Ruta donde se crea el fichero taggedProfileSeq_deco.xml
	 */
	public static final String
TAGGEDPROFILESEQ_DECO="/root/workspace/PIDS-
WS/documentos/intermedios/taggedProfileSeq_deco.xml";

	/**
	 * Ruta donde se crea el fichero traits_known_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
	 */
```

```
	public static final String TRAITS_KNOWN_COD="/root/workspace/PIDS-
WS/documentos/salidas/traits_known_cod.xml";

	/**
	 * Ruta donde se crea el fichero traits_known_deco.xml
	 */
	public static final String TRAITS_KNOWN_DECO="/root/workspace/PIDS-
WS/documentos/intermedios/traits_known_deco.xml";

}
```

14.2 Cliente

Client.java

```
package Client;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import PID.DataTypes.IdStateSeq;

/**
 * Esta clase es el servlet de la aplicación web cliente
 *
 * @author Pilar Mª Juan Montilla
 *
 */
@SuppressWarnings("serial")
public class Client extends HttpServlet{

	public void doGet(HttpServletRequest request, HttpServletResponse
response)
	throws ServletException, IOException
	{
		doPost(request, response);
	}

	public void doPost(HttpServletRequest request, HttpServletResponse
response)
	//throws ServletException, IOException
	{
		String ruta_salida=null;
		//Instanciamos un objeto de cada clase que necesitará el
		//servlet

		IdentifyPersonClient IP= new IdentifyPersonClient();
		ProfileAccessClient PA= new ProfileAccessClient();

		Create_XML XML = new Create_XML ();

		//Si el método es get_traits_known
		if(request.getParameter("metodo").equals("get_traits_known"))
		{
			String id = null;
			id = request.getParameter("identificador");
```

```
                  //Si me indican un ID, llamo al método
                  //get_traist_known_cli
                  if(id != null && id.length()!= 0){

                        System.out.println("identificador: "+id);
                        // ya tenemos todos los datos que necesitamos
                        ruta_salida=PA.get_traits_known_cli(id);

                  }

                  else
                        System.out.println("Error en get_traits_known: No
se ha especificado ningún ID");

            }

            //Si el método es find_candidates
            if(request.getParameter("metodo").equals("find_candidates"))
            {
                  String ruta_profile_selector=null;
                  String ruta_traits_requested=null;
                  String ST_sequence_max=null;

                  //Creamos el fichero de entrada profile_selector.xml
                  ruta_profile_selector=
                        XML.create_profile_selector(request);

                  //Si el formulario no estaba vacío
                  if(ruta_profile_selector!=null)
                  {
                        System.out.println(ruta_profile_selector);
                        //Creamos el fichero de entrada
                        //traits_reuqested.xml
                        ruta_traits_requested=
                              XML.create_traits_requested(request);

                        //Si se seleccionó al menos un trait
                        if(ruta_traits_requested!=null)
                        {
                          System.out.println(ruta_traits_requested);

                          ST_sequence_max=
                              request.getParameter("sequence_max");

                          //Número máximo de candidatos devueltos en
                          //returned_secuence.xml
                          long sequence_max= new
                              Long(ST_sequence_max).longValue();
                          System.out.println("Número máximo de candidatos
devueltos "+sequence_max);
                          //Número maximo de candidatos devueltos en
                          //returned_iterator.xml
                          long iterator_max=10;
                          //Estados admitidos para el identificador de la
                          //persona (no implementado).
                          IdStateSeq states_of_interest=null;
                          //Confianza minima requerida en el candidato
                          //devuelto.
```

```
                        float confidence_threshold=(float)0.50;

                        //ya tenemos todos los datos que necesitamos
                        ruta_salida=
                            IP.find_candidates_cli(ruta_profile_selector,
                            states_of_interest,confidence_threshold,
                            sequence_max,iterator_max,
                            ruta_traits_requested);
                        System.out.println(ruta_salida);

                    }
                     else
                           System.out.println("Error en find_candidates:
No se ha especificado ningún trait\n");
                }
                else
                      System.out.println("Error en find_candidates:
Formulario vacío\n");


            }

            //Si el método es get_profile
            if(request.getParameter("metodo").equals("get_profile")){

                  String ruta_traits_requested=null;
                  String id = null;
                  id = request.getParameter("identificador");

                  //Si me indican un ID, continuo
                  if(id != null && id.length()!= 0){

                        System.out.println("identificador: "+id);
                        //creamos el fichero de entrada
                        //traits_requested.xml
                        ruta_traits_requested=
                              XML.create_traits_requested(request);

                        //Si se seleccionó al menos un trait
                        if (ruta_traits_requested!=null){

                              System.out.println(ruta_traits_requested);
                              //ya tenemos todo los datos que necesitamos
                              ruta_salida=
                              PA.get_profile_cli(id,ruta_traits_requested);
                              System.out.println(ruta_salida);
                        }
                        else
                              System.out.println("Error en get_profile: No
se ha especificado ningún trait\n");
                  }
                  else
                        System.out.println("Error en get_profile: No se ha
indicado ningún ID");
            }

            //Si el método es get_profile_list
```

```
            if(request.getParameter("metodo").equals("get_profile_list")){

                  String ruta_traits_requested=null;
                  String ids []=null;

                  //Obtenemos los ids del formulario
                  ids=obtiene_ids(request);

                  if(ids!=null){

                        //creamos el fichero de entrada
                        //traits_requested.xml
                        ruta_traits_requested=
                              XML.create_traits_requested(request);

                        //Si se seleccionó al menos un trait
                        if (ruta_traits_requested!=null){

                              System.out.println(ruta_traits_requested);
                              //ya tenemos todo los datos que necesitamos
                              ruta_salida=
                        PA.get_profile_list_cli(ids,ruta_traits_requested);
                              System.out.println(ruta_salida);
                        }
                        else
                              System.out.println("Error en get_profile_list
: No se ha especificado ningún trait\n");

                  }
                  else
                        System.out.println("Error en get_profile_list: No
se ha especificado ningún id\n");


            }

            //si todo ha ido bien redireccionamos al fichero de salida
            if (ruta_salida!=null){

                  try{

                        RequestDispatcher dispatcher =
                              request.getRequestDispatcher(ruta_salida);

                        dispatcher.forward(request, response);

                  } catch (ServletException e) {

                        e.printStackTrace();

                  } catch (IOException e) {

                        e.printStackTrace();
                  }

            }
            else
                  System.out.println("No se generó el fichero de salida del
```

```
método invocado");

	}

	/**
	 * Método que recibe la petición HTTP que hace el usuario remoto a
través del formulario
	 * y recupera todos los ids que se han indicado en una tabla de
Strings.
	 *
	 * @param request Es la petición HTTP.
	 * @return Tabla de Strings con todos los ids que se han especificado
en la petición.
	 * en el formulario.
	 */
	public String [] obtiene_ids (HttpServletRequest request){

		String ids []=null;
		int num_ids=0;

		String id1=request.getParameter("id1");
		if(id1!=null && id1.length()!= 0)
			num_ids++;

		String id2=request.getParameter("id2");
		if(id2!=null && id2.length()!= 0)
			num_ids++;

		String id3=request.getParameter("id3");
		if(id3!=null && id3.length()!= 0)
			num_ids++;

		String id4=request.getParameter("id4");
		if(id4!=null && id4.length()!= 0)
			num_ids++;

		String id5=request.getParameter("id5");
		if(id5!=null && id5.length()!= 0)
			num_ids++;

		//Si se ha especificado al menos un id
		if(num_ids!=0){

			System.out.println("Se han indicado "+num_ids+"ids\n");

			ids= new String[num_ids];

			int i=0;

			if(id1!=null && id1.length()!= 0){
				ids[i]=id1;
				i++;
			}

			if(id2!=null && id2.length()!= 0){
				ids[i]=id2;
				i++;
```

```
                }

                if(id3!=null && id3.length()!= 0){
                      ids[i]=id3;
                      i++;
                }

                if(id4!=null && id4.length()!= 0){
                      ids[i]=id4;
                      i++;
                }


                if(id5!=null && id5.length()!= 0)
                      ids[i]=id5;
          }



          return ids;
     }

}
```

Create_XML.java

```
package Client;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.Timestamp;

import javax.servlet.http.HttpServletRequest;

import org.exolab.castor.mapping.Mapping;
import org.exolab.castor.mapping.MappingException;
import org.exolab.castor.xml.MarshalException;
import org.exolab.castor.xml.Marshaller;
import org.exolab.castor.xml.ValidationException;

import GPICS.LanguageCommunication;
import GPICS.CAG_GPICS.EntityName;
import GPICS.CAG_GPICS.EntityNamePart;
import GPICS.CAG_GPICS.PostalAddress;
import GPICS.CAG_GPICS.PostalAddressPart;
import PID.DataTypes.SpecifiedTraits;
import PID.DataTypes.Trait;
import PID.DataTypes.TraitSelector;
import PID.DataTypes.TraitSelectorSeq;
import PID.DataTypes.TraitSpec;
import basicTypes.BL;
import basicTypes.CS;
import basicTypes.CV;
```

```
import basicTypes.II;
import basicTypes.IVL_TS;
import basicTypes.List;
import basicTypes.ST;
import basicTypes.TS;
import basicTypes.URL;

/**
 * Clase que se encarga de crear los ficheros profile_selector.xml y
traits_requested.xml a partir de la
 * información recuperada de los formularios.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class Create_XML {



      /**
       * Método que se encarga de crear el fichero profile_selector.xml a
partir de la información recuperada de la petición HTTP.
       *
       * @param request Petición HTTP que realiza el usuario remoto a
través de los formularios.
       * @return Ruta absoluta del fichero profile_selector.xml
       */
      public String create_profile_selector (HttpServletRequest request){

            String ruta_profile_selector=null; //lo que devolverá el método
            List entityNameValue=new List();
            List name=new List();
            List dire=new List();
            PostalAddressPart dir1=null;//provincia
            PostalAddressPart dir2=null;//localidad
            PostalAddressPart dir_solo=null;
            PostalAddress direccion1=null;
            ST cod_postal_solo=null;
            II identificadorDNI=null;
            URL url=null;
            LanguageCommunication language=null;
            CV nacion1=null;
            List nacionalidades=new List();
            CV religeouscode=null;

            float weight=(float)0.50;
            boolean relleno=false;
            int num_traits=0;
            boolean one_dir=false;

            //Variables donde recuperaremos la información del formulario
            String nombre=null;
            String apellido1=null;
            String localidad=null;
            String provincia=null;
            String codigopostal=null;
            String dni=null;
```

```
            String telefono=null;
            String idioma=null;
            String nacionalidad=null;
            String religion=null;

            TS i = new TS(Timestamp.valueOf("1980-08-22 00:00:00"));
            TS f = new TS(Timestamp.valueOf("2015-08-22 00:00:00"));
            IVL_TS validTime = new IVL_TS(i,f,new BL(true),new BL(true),
            Long.toString(f.getTimePoint().getTime()-
i.getTimePoint().getTime()));


          //Recopilamos la información recogida del formulario para crear
          //el fichero profile_selector.xml

            nombre=request.getParameter("nombre");

            if(nombre!=null && nombre.length()!=0){

                  relleno=true;
                  EntityNamePart nombrep1=new EntityNamePart(new
ST(nombre,new CS(), new CS(new ST("espanol",null,null),null)), new CS(new
ST("nombre propio",null,null), new ST("espanol",null,null)) , new CS(new
ST("preferido",null,null),new ST("espanol",null,null)));


                  apellido1=request.getParameter("apellido1");

                  if(apellido1!=null && apellido1.length()!=0){

                        //SE ESPECIFICA NOMBRE Y APELLIDO


                        EntityNamePart nombrep2=new EntityNamePart(new
ST(apellido1,new CS(),new CS(new ST("espanol",null,null),null)),new CS(new
ST("apellido",null,null),new ST("espanol",null,null)), new CS(new
ST("preferido",null,null),new ST("espanol",null,null)));


                        entityNameValue.add(nombrep1);
                        entityNameValue.add(nombrep2);

                        EntityName nombre_completo= new
                              EntityName(entityNameValue, validTime);

                        num_traits++;
                        name.add(nombre_completo);
                  }
                  else{

                        //SE HA ESPECIFICADO SÓLO EL NOMBRE

                        System.out.println("No se ha indicado el primer
apellido\n");
                        entityNameValue.add(nombrep1);
                        EntityName nombre_completo=new
                              EntityName(entityNameValue, validTime);
```

```
                        num_traits++;
                        name.add(nombre_completo);
                  }
            }
            else{

                  apellido1=request.getParameter("apellido1");

                  if(apellido1!=null && apellido1.length()!=0){

                        //SE HA ESPECIFICADO SÓLO EL PRIMER APELLIDO

                        relleno=true;
                        System.out.println("Solo se ha especificado el
apellido\n");
                        EntityNamePart nombrep2=new EntityNamePart(new
      ST(apellido1,new CS(),new CS(new ST("espanol",null,null),null)),
      new CS(new ST("apellido",null,null),new ST("espanol",null,null)),
      new CS(new ST("preferido",null,null),new ST("espanol",null,null)));

                        entityNameValue.add(nombrep2);

                        EntityName nombre_completo=new
                              EntityName(entityNameValue,validTime);

                        num_traits++;
                        name.add(nombre_completo);


                  }
                  else{
                        System.out.println("No se ha especificado ni el
nombre ni el apellido\n");
                  }


            }

            localidad=request.getParameter("localidad");

            if(localidad!=null && localidad.length()!=0){

                  relleno=true;

                  dir2=new PostalAddressPart(new ST(localidad,new CS(),
                        new CS(new ST("espanol",null,null),null)),
                        new CS(new ST("CTY",null,null),new
                        ST("espanol",null,null)));

                  provincia=request.getParameter("provincia");

                  if(provincia!=null && provincia.length()!=0){

                        dir1=new PostalAddressPart(new ST(provincia,
new CS(),new CS(new ST("espanol",null,null),null)), new CS(new
ST("CNT",null,null),new ST("espanol",null,null)));
```

```
                        dire.add(dir1);
                        dire.add(dir2);

                  codigopostal=request.getParameter("codigopostal");

                  if(codigopostal!=null && codigopostal.length()!=0){

                        //SE HA ESPECIFICADO LOCALIDAD, PROVINCIA Y CÓDIGO
                        //POSTAL
                        direccion1=new PostalAddress(dire, new
                              ST(codigopostal,null,null), null,validTime);
                        num_traits++;
                  }
                  else{
                        //SE HA ESPECIFICADO LOCALIDAD Y PROVINCIA

                        direccion1=new
                              PostalAddress(dire,null,null,validTime);
                        num_traits++;
                  }

            }
            else{

                  codigopostal=request.getParameter("codigopostal");

                  if(codigopostal!=null && codigopostal.length()!=0){

                        //SE HA ESPECIFICADO LA LOCALIDAD Y EL CÓDIGO
                        //POSTAL

                        dire.add(dir2);
                        direccion1=new PostalAddress(dire, new
                              ST(codigopostal,null,null),null,validTime);
                        num_traits++;

                  }
                  else{

                        //SÓLO SE HA ESPECIFICADO LA LOCALIDAD
                        one_dir=true;
                        dir_solo=dir2;
                        num_traits++;

                  }
            }

      }
      else{

            provincia=request.getParameter("provincia");

            if(provincia!=null && provincia.length()!=0){

                  relleno=true;

                  dir1=new PostalAddressPart(new ST(provincia,
new CS(),new CS(new ST("espanol",null,null),null)),new CS(new
```

```
ST("CNT",null,null),new ST("espanol",null,null)));

                  codigopostal=request.getParameter("codigopostal");

                  if(codigopostal!=null && codigopostal.length()!=0){

                        //SE HA ESPECIFICADO LA PROVINCIA Y EL CODIGO
                        //POSTAL
                        dire.add(dir1);
                        direccion1=new PostalAddress(dire,
                              new ST(codigopostal,null,null),
                              null,validTime);
                        num_traits++;
                  }
                  else{
                        //SÓLO SE HA ESPECIFICADO LA PROVINCIA
                        one_dir=true;
                        dir_solo=dir1;
                        num_traits++;
                  }



            }
            else{

                  codigopostal=request.getParameter("codigopostal");

                  if(codigopostal!=null && codigopostal.length()!=0){

                        //SÓLO SE HA ESPECIFICADO EL CÓDIGO POSTAL
                        relleno=true;

                        one_dir=true;
                        cod_postal_solo=new ST(codigopostal,null,null);
                        num_traits++;
                  }
                   else
                        System.out.println("No se ha especificado ningún
componente de dirección");

                  }


            }

            dni=request.getParameter("dni");

            if(dni!=null && dni.length()!=0){

                  //SE ESPECIFICA EL DNI

                  relleno=true;

                  identificadorDNI=new II(new ST(dni,null,null),null,
                        new CS(new ST("DNI",null,null),null),null);
```

```
                num_traits++;

            }
            else
                System.out.println("No se ha indicado el DNI\n");


            telefono=request.getParameter("telefono");

            if(telefono!=null && telefono.length()!=0){

                // SE ESPECIFICA EL NÚMERO DE TELÉFONO

                relleno=true;

                url=new URL( new ST(telefono,null,null),
                    null,null,null,null,null);

                num_traits++;

            }
            else
                System.out.println("No se ha indicado el número de
teléfono\n");


            idioma=request.getParameter("idioma");

            if(idioma!=null && idioma.length()!=0){

                // SE ESPECIFICA EL IDIOMA

                relleno=true;

                language= new LanguageCommunication(new CV(
                    new ST(idioma),null,null,null,null,null),
                    null,null,null);

                num_traits++;
            }
            else
                System.out.println("No se ha indicado el idioma\n");

            nacionalidad=request.getParameter("nacionalidad");

            if(nacionalidad!=null && nacionalidad.length()!=0){

                // SE ESPECIFICA LA NACIONALIDAD

                relleno=true;

                nacion1=new CV(new
                    ST(nacionalidad,null,null),null,null,null,
                    null,null);
                nacionalidades.add(nacion1);

                num_traits++;
            }
```

```
            else
                  System.out.println("No se ha indicado la
nacionalidad\n");

            religion=request.getParameter("religion");

            if(religion!=null && religion.length()!=0){

                  // SE ESPECIFICA LA RELIGIÓN

                  relleno=true;

                  religeouscode= new CV(new
                        ST(religion,null,null),null,null,null,null,null);
                  num_traits++;
            }
            else
                  System.out.println("No se ha indicado la religión\n");



            //Si se ha rellenado al menos un campo del formulario creamos
            //el fichero de entrada profile_selector.xml
            if (relleno){

                   TraitSelectorSeq profile_selector=new
                        TraitSelectorSeq(num_traits);
                   int indice=0;

                   if((nombre!=null || apellido1!=null) &&
                  (nombre.length()!=0 || apellido1.length()!=0)){
                         Trait trait1=new Trait("name", name);
                         profile_selector.getSeq()[indice]=
                              new TraitSelector(trait1, weight);
                         indice++;
                   }

                   if (dni!=null && dni.length()!=0){
                         Trait trait2=new Trait("II", identificadorDNI);
                         profile_selector.getSeq()[indice]=
                              new TraitSelector(trait2, weight);
                         indice++;
                   }

                   if(telefono!=null && telefono.length()!=0){
                         Trait trait3=new Trait("URL", url);
                         profile_selector.getSeq()[indice]=
                              new TraitSelector(trait3, weight);
                         indice++;
                   }

                   if(idioma!=null && idioma.length()!=0){
                         Trait trait4=
                              new Trait("LanguageCommunication", language);
                         profile_selector.getSeq()[indice]=
                              new TraitSelector(trait4, weight);
                         indice++;
```

```
                    }

                    if(nacionalidad!=null && nacionalidad.length()!=0){
                          Trait trait5=new Trait("nationalityCode",
                               nacionalidades);
                          profile_selector.getSeq()[indice]=
                               new TraitSelector(trait5, weight);
                          indice++;
                    }

                    if(religion!=null && religion.length()!=0){
                          Trait trait6=
                               new Trait("religeousAffiliationCode",
                               religeouscode);
                          profile_selector.getSeq()[indice]=
                               new TraitSelector(trait6, weight);
                          indice++;
                    }


                    if((localidad!=null || provincia!=null) &&
(localidad.length()!=0 || provincia.length()!=0)&& (!one_dir)){

                          // más de un componente de dirección
                          Trait trait7=
                               new Trait("PostalAddress", direccion1);
                          profile_selector.getSeq()[indice]=
                               new TraitSelector(trait7, weight);
                          indice++;

                    }
                    else{

                          if(one_dir){

                                //sólo un componente de dirección

                                if (cod_postal_solo!=null){
                                      Trait trait7=
                               new Trait("addressLine", cod_postal_solo);
                                      profile_selector.getSeq()[indice]=
                                     new TraitSelector(trait7, weight);
                                      indice++;
                                }
                                else{

                                      Trait trait7=
                               new Trait("PostalAddressPart", dir_solo);
                                      profile_selector.getSeq()[indice]=
                                     new TraitSelector(trait7, weight);
                                      indice++;

                                }

                          }

                    }
```

```
                  //Pasar todo lo anterior a archivos para poder
                  //imitar a como se recibirian por la red
                  //Creamos el mapping y lo cargamos en su controlador
                  Mapping mapa= new Mapping();

                  try {

                        mapa.loadMapping(Constantes.PIDS);

                        //Creamos el elemento encargado de transformar de
                        //objeto a XML
                        Marshaller m=null;

                        System.out.println("Creamos el fichero
profile_selector.xml");

                  ruta_profile_selector=Constantes.PROFILE_SELECTOR;

                  File file_profile_selector=
                        new File(ruta_profile_selector);

                  FileWriter file_writer_profile_selector = null;

                  file_writer_profile_selector =
                        new FileWriter(file_profile_selector);

                  m = new Marshaller(file_writer_profile_selector);

                  m.setMapping(mapa);

                  m.marshal(profile_selector);

                  file_writer_profile_selector.close();


                  }catch (IOException e){

                   e.printStackTrace();

                  }catch (MappingException e){

                    e.printStackTrace();

                  }catch (ValidationException e){

                    e.printStackTrace();
                  }catch (MarshalException e) {

                              e.printStackTrace();
                  }

            }


            return ruta_profile_selector;

      }
```

```
	/**
	 * Método que se encarga de crear el fichero traits_requested.xml a
partir de la información recuperada de la petición HTTP.
	 *
	 * @param request Petición HTTP que realiza el usuario remoto a
través de los formularios.
	 * @return Ruta absoluta del fichero traits_requested.xml
	 */
	public String create_traits_requested (HttpServletRequest request){

		String ruta_traits_requested=null; //lo que devolverá el método
		String b=null;
		int num_traits=0; //Para ir contando el número de traits
		                  //solicitados

		//Atributos solicitados de cada candidato devuelto
		SpecifiedTraits traits_requested=new SpecifiedTraits();

		//Primero contamos traits se han solicitado

		b=request.getParameter("addr");

		if(b!=null){
			num_traits++;
			System.out.println(b);
			b=null;
		}

		b=request.getParameter("administrativeGenderCode");

		if(b!=null){
			num_traits++;
			System.out.println(b);
			b=null;
		}

		b=request.getParameter("birthTime");

		if(b!=null){
			num_traits++;
			System.out.println(b);
			b=null;
		}

		b=request.getParameter("birthOrderNumber");

		if(b!=null){
			num_traits++;
			System.out.println(b);
			b=null;
		}

		b=request.getParameter("code");

		if(b!=null){
			num_traits++;
			System.out.println(b);
```

```
		b=null;
	}

	b=request.getParameter("deceasedInd");

	if(b!=null){
		num_traits++;
		System.out.println(b);
		b=null;
	}

	b=request.getParameter("deceasedTime");

	if(b!=null){
		num_traits++;
		System.out.println(b);
		b=null;
	}

	b=request.getParameter("disabilityCode");

	if(b!=null){
		num_traits++;
		System.out.println(b);
		b=null;
	}

	b=request.getParameter("employmentStatusCode");

	if(b!=null){
		num_traits++;
		System.out.println(b);
		b=null;
	}

	b=request.getParameter("ethnicGroupCode");

	if(b!=null){
		num_traits++;
		System.out.println(b);
		b=null;
	}

	b=request.getParameter("healthcareProfessionalRole");

	if(b!=null){
		num_traits++;
		System.out.println(b);
		b=null;
	}

	b=request.getParameter("id");

	if(b!=null){
		num_traits++;
		System.out.println(b);
		b=null;
```

```
}

b=request.getParameter("jobCode");

if(b!=null){
      num_traits++;
      System.out.println(b);
      b=null;
}

b=request.getParameter("jobTitleCode");

if(b!=null){
      num_traits++;
      System.out.println(b);
      b=null;
}

b=request.getParameter("languageCommunication");

if(b!=null){
      num_traits++;
      System.out.println(b);
      b=null;
}

b=request.getParameter("livingArrangmentCode");

if(b!=null){
      num_traits++;
      System.out.println(b);
      b=null;
}

b=request.getParameter("name");

if(b!=null){
      num_traits++;
      System.out.println(b);
      b=null;
}

b=request.getParameter("nationalityCode");

if(b!=null){
      num_traits++;
      System.out.println(b);
      b=null;
}

b=request.getParameter("maritalStatusCode");

if(b!=null){
      num_traits++;
      System.out.println(b);
      b=null;
}
```

```
            b=request.getParameter("patientExtendedInformation");

            if(b!=null){
                  num_traits++;
                  System.out.println(b);
                  b=null;
            }

            b=request.getParameter("patientStandardInformation");

            if(b!=null){
                  num_traits++;
                  System.out.println(b);
                  b=null;
            }

            b=request.getParameter("person");

            if(b!=null){
                  num_traits++;
                  System.out.println(b);
                  b=null;
            }

            b=request.getParameter("religeousAffiliationCode");

            if(b!=null){
                  num_traits++;
                  System.out.println(b);
                  b=null;
            }

            b=request.getParameter("riskCode");

            if(b!=null){
                  num_traits++;
                  System.out.println(b);
                  b=null;
            }

            b=request.getParameter("telecom");

            if(b!=null){
                  num_traits++;
                  System.out.println(b);
                  b=null;
            }

            if(num_traits==0)
                  System.out.println("Error en create_traits_requested: No
se ha pedido ningún trait");
            else {

                  System.out.println("\nSe han solicitado "+num_traits+"
traits\n");

                  //Vamos añadiendo traits a traits_requested
```

```
traits_requested.setSeq(new TraitSpec[num_traits]);

int i=0;
b=request.getParameter("addr");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("addr", false, true, false);
      System.out.println("addr");
      i++;
      b=null;
}

b=request.getParameter("administrativeGenderCode");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("administrativeGenderCode",
            false, true, false);
      System.out.println("administrativeGenderCode");
      i++;
      b=null;
}

b=request.getParameter("birthTime");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("birthTime",false,true,false);
      System.out.println("birthTime");
      i++;
      b=null;
}

b=request.getParameter("birthOrderNumber");

if(b!=null){
      traits_requested.getSeq()[i]=
      new TraitSpec("birthOrderNumber",false,true,false);
      System.out.println("birthOrderNumber");
      i++;
      b=null;
}

b=request.getParameter("code");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("code", false, true, false);
      System.out.println("code");
      i++;
      b=null;
}

b=request.getParameter("deceasedInd");

if(b!=null){
```

```
        traits_requested.getSeq()[i]=
        new TraitSpec("deceasedInd",false,true,false);
        System.out.println("deceasedInd");
        i++;
        b=null;
}

b=request.getParameter("deceasedTime");

if(b!=null){
        traits_requested.getSeq()[i]=
        new TraitSpec("deceasedTime",false,true,false);
        System.out.println("deceasedTime");
        i++;
        b=null;
}

b=request.getParameter("disabilityCode");

if(b!=null){
        traits_requested.getSeq()[i]=
        new TraitSpec("disabilityCode",false,true,false);
        System.out.println("disabilityCode");
        i++;
        b=null;
}

b=request.getParameter("employmentStatusCode");

if(b!=null){
        traits_requested.getSeq()[i]=
              new TraitSpec("employmentStatusCode",false,
              true, false);
        System.out.println("employmentStatusCode");
        i++;
        b=null;
}

b=request.getParameter("ethnicGroupCode");

if(b!=null){
        traits_requested.getSeq()[i]=
        new TraitSpec("ethnicGroupCode", false,
         true, false);
        System.out.println("ethnicGroupCode");
        i++;
        b=null;
}

b=request.getParameter("healthcareProfessionalRole");

if(b!=null){
        traits_requested.getSeq()[i]=
              new TraitSpec("healthcareProfessionalRole",
              false, true, false);
        System.out.println("healthcareProfessionalRole");
        i++;
```

```
      b=null;
}

b=request.getParameter("id");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("id", false, true, false);
      System.out.println("id");
      i++;
      b=null;
}

b=request.getParameter("jobCode");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("jobCode", false, true, false);
      System.out.println("jobCode");
      i++;
      b=null;
}

b=request.getParameter("jobTitleCode");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("jobTitleCode",false,
            true, false);
      System.out.println("jobTitleCode");
      i++;
      b=null;
}

b=request.getParameter("languageCommunication");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("languageCommunication",false,
            true, false);
      System.out.println("languageCommunication");
      i++;
      b=null;
}

b=request.getParameter("livingArrangmentCode");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("livingArrangmentCode", false,
            true, false);
      System.out.println("livingArrangmentCode");
      i++;
      b=null;
}

b=request.getParameter("name");
```

```
if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("name", false, true, false);
      System.out.println("name");
      i++;
      b=null;
}

b=request.getParameter("nationalityCode");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("nationalityCode",false,
            true, false);
      System.out.println("nationalityCode");
      i++;
      b=null;
}

b=request.getParameter("maritalStatusCode");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("maritalStatusCode",false,
            true, false);
      System.out.println("maritalStatusCode");
      i++;
      b=null;
}

b=request.getParameter("patientExtendedInformation");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("patientExtendedInformation",
            false, true, false);
      System.out.println("patientExtendedInformation");
      i++;
      b=null;
}

b=request.getParameter("patientStandardInformation");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("patientStandardInformation",
            false, true, false);
      System.out.println("patientStandardInformation");
      i++;
      b=null;
}

b=request.getParameter("person");

if(b!=null){
      traits_requested.getSeq()[i]=
            new TraitSpec("person", false, true, false);
```

```
                        System.out.println("person");
                        i++;
                        b=null;
                  }

                  b=request.getParameter("religeousAffiliationCode");

                  if(b!=null){
                        traits_requested.getSeq()[i]=
                              new TraitSpec("religeousAffiliationCode",
                              false, true, false);
                        System.out.println("religeousAffiliationCode");
                        i++;
                        b=null;
                  }

                  b=request.getParameter("riskCode");

                  if(b!=null){
                        traits_requested.getSeq()[i]=
                        new TraitSpec("riskCode", false, true, false);
                        System.out.println("riskCode");
                        i++;
                        b=null;
                  }

                  b=request.getParameter("telecom");

                  if(b!=null){
                        traits_requested.getSeq()[i]=
                              new TraitSpec("telecom", false, true, false);
                        System.out.println("telecom");
                        b=null;
                  }

                  System.out.println("Preparo el archivo de entrada
traits_requested.xml");

                  try{


                  Mapping mapa= new Mapping();

                  mapa.loadMapping(Constantes.PIDS);

                //Creamos el elemento encargado de transformar de objeto a
                //XML
                Marshaller m=null;

                ruta_traits_requested=Constantes.TRAITS_REQUESTED;
                File file_traits_requested =
                  new File(ruta_traits_requested);
                FileWriter file_writer_traits_requested=
                  new FileWriter(file_traits_requested);

                m = new Marshaller(file_writer_traits_requested);
                m.setMapping(mapa);
                m.marshal(traits_requested);
```

```
                file_writer_traits_requested.close();


                  }catch (IOException e){
                  System.out.println("IOException: \n");
                  e.printStackTrace();

                 }catch (MappingException e){

                   System.out.println("MappingException: \n");
                   e.printStackTrace();

                 }catch (ValidationException e){

                   System.out.println("ValidationException: \n");
                   e.printStackTrace();

                 }catch (MarshalException e) {

                  System.out.println("MarshalException: \n");
                  e.printStackTrace();
                 }


            }


            return ruta_traits_requested;
      }

}
```

IdentifyPersonClient.java

```
package Client;

import java.io.File;
import java.io.IOException;

import javax.xml.stream.XMLStreamException;

import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;
import org.apache.axis2.AxisFault;
import org.apache.axis2.Constants;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;

import PID.DataTypes.IdStateSeq;
import Codification.Canonizador;
```

```
/**
 * Clase que se encarga de invocar al método find_candidates de la interfaz
IdentifyPerson del Servicio Web
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class IdentifyPersonClient {

      static EndpointReference targetEPR =
            new
EndpointReference("http://localhost:8080/axis2/services/IdentifyPerson");

      static OMFactory fac = OMAbstractFactory.getOMFactory();
      static OMNamespace omNs =
            fac.createOMNamespace("http://service/xsd","IdentifyPerson");


      /**
       * Método que recibe todos los datos recopilados de los formularios
de entrada para invocar al método find_candidates de la interfaz
IdentifyPerson del Servicio Web.
       *
       * @param ruta_profile_selector Ruta del fichero de entrada
profile_selector.xml.
       * @param states_of_interest Estados admitidos para el identificador
de la persona (no implementado).
       * @param confidence_threshold Confianza minima requerida en el
candidato devuelto.
       * @param sequence_max Número máximo de candidatos devueltos en
returned_secuence.xml
       * @param iterator_max Número maximo de candidatos devueltos en
returned_iterator.xml
       * @param ruta_traits_requested Ruta del fichero de entrada
traits_requested.xml.
       * @return Ruta relativa del fichero de salida.
       */
      public String find_candidates_cli(String ruta_profile_selector,
IdStateSeq states_of_interest, float confidence_threshold, long
sequence_max, long iterator_max, String ruta_traits_requested){

            OMElement profile_selector_cod=null;
            OMElement traits_requested_cod=null;
            String ruta_traits_requested_cod=null;
            String ruta_profile_selector_cod=null;
            String ruta_returned_sequence_cod=null;
            String ruta_returned_iterator_cod=null;
            String ruta_returned_sequence=null;
            String ruta_returned_iterator=null;
            String contenido_profile_selector_cod = null;
            String contenido_traits_requested_cod = null;
            String ruta_relativa_salida=null;

           //PREPARAMOS traits_requested_cod.xml (sin acentos)

           //Creamos el archivo de entrada que se supone que recibe el
```

```
            //programa con tildes
            File file_profile_selector = new File(ruta_profile_selector);

            //Creamos el codificador para normalizar el XML
            Canonizador canonizador=new Canonizador();

            //Creeamos el archivo de entrada que recibe el método sin
            //acentos
            ruta_profile_selector_cod=Constantes.PROFILE_SELECTOR_COD;
            File file_profile_selector_cod=
                  new File(ruta_profile_selector_cod);

             try {
                   //Se rellena el archivo sin acentos
                   canonizador.canoniza(file_profile_selector,
                         file_profile_selector_cod);
             } catch (IOException e1) {

                  e1.printStackTrace();
            }

            //PREPARAMOS traits_reuqested_cod.xml (sin acentos)

            //Creamos el archivo de entrada que se supone que recibe el
            //programa con tildes
            File file_traits_requested = new File(ruta_traits_requested);


            //Creamos el archivo de entrada que recibe el método sin
            //acentos
            ruta_traits_requested_cod=Constantes.TRAITS_REQUESTED_COD;
            File file_traits_requested_cod=
                  new File(ruta_traits_requested_cod);

            //Rellenamos el archivo sin acentos
            try {
                  canonizador.canoniza(file_traits_requested,
                        file_traits_requested_cod);
            } catch (IOException e1) {

                  e1.printStackTrace();
            }

            //CREAMOS LOS OBJETOS OMELEMENT CORRESPONDIENTES A LOS ficheros
            //profile_selector_cod.xml y traits_requested_cod.xml
            profile_selector_cod=
                  Auxiliar.get_OMElement(ruta_profile_selector_cod);
            traits_requested_cod=
                  Auxiliar.get_OMElement(ruta_traits_requested_cod);

            try {

                  contenido_profile_selector_cod =
                        profile_selector_cod.toStringWithConsume();
                  System.out.println("El contenido del fichero
profile_selector_cod.xml es el siguiente: \n"
+contenido_profile_selector_cod);
```

```
                  contenido_traits_requested_cod =
                        traits_requested_cod.toStringWithConsume();
                  System.out.println("El contenido del fichero
traits_requested_cod.xml es el siguiente: \n"
+contenido_traits_requested_cod);

            } catch (XMLStreamException e) {

              System.out.println("Se ha producido el siguiente error 1:\n");
              e.printStackTrace();
            }


            //PREPARAMOS EL OMELEMENT CON LA PETICIÓN AL SERVIDOR
            OMElement peticion=
                  crea_OMElement_PI(contenido_profile_selector_cod,
                  states_of_interest, confidence_threshold,sequence_max,
                  iterator_max,contenido_traits_requested_cod);

            //PREPARAMOS LA PETICIÓN AL SERVIDOR
            Options options = new Options();
            options.setTo(targetEPR);
            options.setTransportInProtocol(Constants.TRANSPORT_HTTP);

            ServiceClient sender;
            OMElement resultado=null;

             try {

                 sender = new ServiceClient();

                 sender.setOptions(options);

                 System.out.println("Invocamos al servicio IdentifyPerson");

                 resultado =sender.sendReceive(peticion);


             } catch (AxisFault e) {

                   System.out.println("Error al crear el ServiceClient");
                   e.printStackTrace();

             }

             //RECUPERAMOS LO QUE NOS DEVUELVE EL SERVICIO Y CREAMOS LOS
             //DOS FICHEROS XML DE SALIDA:
             // returned_sequence.xml y returned_iterator.xml
             if (resultado!=null)
             {

                   OMElement returned_sequence_cod=
                         resultado.getFirstElement();
                   String contenido_returned_sequence_cod=
                         returned_sequence_cod.getText();

                   OMElement returned_iterator_cod= (OMElement)
```

```
                    returned_sequence_cod.getNextOMSibling();
                String contenido_returned_iterator_cod =
                    returned_iterator_cod.getText();

                System.out.println("El contenido del fichero
returned_sequence_cod.xml es el siguiente:
\n"+contenido_returned_sequence_cod);

              System.out.println("El contenido del fichero
returned_iterator_cod.xml es el siguiente:
\n"+contenido_returned_iterator_cod);

                //CREAMOS LOS FICHEROS DE SALIDA
                //returned_sequence_cod.xml y returned_iterator_cod.xml
                //(sin acentos)
                ruta_returned_sequence_cod=
                    Constantes.RETURNED_SEQUENCE_COD;
                ruta_returned_iterator_cod=
                    Constantes.RETURNED_ITERATOR_COD;
                Auxiliar.crea_xml(contenido_returned_sequence_cod,
                    ruta_returned_sequence_cod);
                Auxiliar.crea_xml(contenido_returned_iterator_cod,
                    ruta_returned_iterator_cod);

                //PREPARARAMOS returned_sequence.xml (con acentos)

                //Creamos el archivo de salida que se supone que devuelve
                //el método sin acentos
                File file_returned_sequence_cod =
                    new File(ruta_returned_sequence_cod);


                //Creamos el archivo de salida del método pero con
                //acentos
                ruta_returned_sequence=Constantes.RETURNED_SEQUENCE;
                ruta_relativa_salida=
                    "documentos/salidas/returned_sequence.xml";
                File file_returned_sequence=
                    new File(ruta_returned_sequence);

                try {
                    //Rellenamos el archivo con acentos

                    canonizador.descanoniza(file_returned_sequence_cod,
                          file_returned_sequence);
                 } catch (IOException e1) {

                    e1.printStackTrace();
                 }

                //PREPARAMOS returned_iterator.xml (con acentos)

                //Creamos el archivo de salida que se supone que devuelve
                //el método sin acentos
                File file_returned_iterator_cod =
                    new File(ruta_returned_iterator_cod);
```

```
                  //Creamos el archivo de salida del método pero con
                  //acentos
                  ruta_returned_iterator=Constantes.RETURNED_ITERATOR;
                  File file_returned_iterator=
                        new File(ruta_returned_iterator);

                  try {
                        //Rellenamos el archivo con acentos
                        canonizador.descanoniza(file_returned_iterator_cod,
                              file_returned_iterator);
                  } catch (IOException e1) {

                        e1.printStackTrace();
                  }


            }
            else
                  System.out.println("El servicio no me ha contestado
nada");


            return ruta_relativa_salida;

      }


      /**
       * Método que recibe todos los parámetros que se necesitan para
invocar al método fin_candidates del Servicio Web y se encarga de preparar
el OMElement de petición.
       *
       * @param contenido_profile_selector String con el contenido del
fichero de entrada profile_selector_cod.xml
       * @param states_of_interest_real
       * @param confidence_threshold_real
       * @param sequence_max_real Long con el número máximo de candidatos
devueltos en returned_secuence.xml
       * @param iterator_max_real Long con el número máximo de candidatos
devueltos en returned_iterator.xml
       * @param contenido_traits_requested String con el contenido del
fichero de entrada traits_requested_cod.xml
       * @return OMElement con la petición para invocar al método
find_candidates del Servicio Web.
       */
      public OMElement crea_OMElement_PI (String
contenido_profile_selector, IdStateSeq states_of_interest_real,float
confidence_threshold_real, long sequence_max_real,
long iterator_max_real, String contenido_traits_requested){


            OMElement salida = fac.createOMElement("find_candidates",
                  omNs);

            OMElement value1 = fac.createOMElement("profile_selector",
                  omNs);
            value1.addChild(fac.createOMText(value1,
```

```
                  contenido_profile_selector));
            salida.addChild(value1);

            OMElement value2=fac.createOMElement("states_of_interest",
                  omNs);
            String states_of_interest="";

            if(states_of_interest_real==null){

                  states_of_interest="null";


      System.out.println("states_of_interest:"+states_of_interest);
                  value2.addChild(fac.createOMText(value2,
                        states_of_interest));
                  salida.addChild(value2);
            }
            else
                  System.out.println("Error al especificar states_of
interest");

            OMElement value3 = fac.createOMElement("confidence_threshold",
omNs);
            String confidence_threshold;
            confidence_threshold=""+ confidence_threshold_real;

      System.out.println("confidence_threshold:"+confidence_threshold);
            value3.addChild(fac.createOMText(value3,
                  confidence_threshold));
            salida.addChild(value3);

            OMElement value4 = fac.createOMElement("sequence_max", omNs);
            String sequence_max;
            sequence_max=""+ sequence_max_real;
            System.out.println("sequence_max:"+sequence_max);
            value4.addChild(fac.createOMText(value4, sequence_max));
            salida.addChild(value4);

            OMElement value5 = fac.createOMElement("iterator_max", omNs);
            String iterator_max;
            iterator_max=""+ iterator_max_real;
            System.out.println("iterator_max:"+iterator_max);
            value5.addChild(fac.createOMText(value5, iterator_max));
            salida.addChild(value5);

            OMElement value6=fac.createOMElement("traits_requested",omNs);
            value6.addChild(fac.createOMText(value6,
                  contenido_traits_requested));
            salida.addChild(value6);

            return salida;

      }

}
```

ProfileAccessClient.java

```
package Client;

import java.io.File;
import java.io.IOException;

import javax.xml.stream.XMLStreamException;

import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMFactory;
import org.apache.axiom.om.OMNamespace;

import org.apache.axis2.AxisFault;
import org.apache.axis2.Constants;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;

import Codification.Canonizador;


/**
 * Clase que invoca a los métodos get_traits_known, get_profile y
get_profile_list de la interfaz ProfileAccess del Servicio Web
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class ProfileAccessClient {

      static EndpointReference targetEPR = new
EndpointReference("http://localhost:8080/axis2/services/ProfileAccess");

      static OMFactory fac = OMAbstractFactory.getOMFactory();
      static OMNamespace omNs =
            fac.createOMNamespace("http://service/xsd","ProfileAccess");


      /**
       * Método que recibe todos los datos necesarios para invocar al
método get_traits_known de la interfaz ProfileAccess del Servicio Web.
       *
       * @param id String que contiene el identificador de una persona.
       * @return Ruta relativa del fichero de salida traits_known.xml.
       */
      public String get_traits_known_cli(String id) {

            String ruta_traits_known=null;
            String ruta_traits_known_cod=null;
            String ruta_relativa_salida=null;

            //PROBAMOS EL MÉTODO get_traits_known
```

```
            //Preparamos el OMElement que indicará al servicio el ID
            OMElement OM_id= fac.createOMElement("get_traits_known", omNs);
            OMElement value = fac.createOMElement("id", omNs);

            value.addChild(fac.createOMText(value, id));
            OM_id.addChild(value);

            //Preparamos la llamada al servicio
            Options options = new Options();
            options.setTo(targetEPR);
            options.setTransportInProtocol(Constants.TRANSPORT_HTTP);


            ServiceClient sender;
            //OMELement que recibiremos del servicio y del que obtendremos
            //el fichero traits_known.xml
            OMElement resultado=null;

            try {

                sender = new ServiceClient();

                sender.setOptions(options);

                System.out.println("Invocamos al servicio ProfileAccess:
get_traist_known");

                resultado =sender.sendReceive(OM_id);

            } catch (AxisFault e) {

                  System.out.println("Error al crear el ServiceClient");
                  e.printStackTrace();

            }

            if (resultado!=null){

                  //RECUPERAMOS EL FICHERO traits_known_cod.xml QUE NOS
                  //DECUELVE EL SERVICIO

                  OMElement traits_known_cod= resultado.getFirstElement();
                  String contenido_traits_known_cod=
                        traits_known_cod.getText();

                  System.out.println("El contenido del fichero
traits_known_cod.xml es el siguiente:\n"+contenido_traits_known_cod);

                  //CREAMOS EL FICHERO traits_known_cod.xml
                  ruta_traits_known_cod=Constantes.TRAITS_KNOWN_COD;
                  Auxiliar.crea_xml(contenido_traits_known_cod,
                        ruta_traits_known_cod);

                  //PREPARAMOS traits_known.xml (con acentos)

                  //Creo el archivo de salida que se supone que devuelve el
                  //método sin acentos
```

```
                File file_traits_known_cod =
                      new File(ruta_traits_known_cod);


                //Creamos el archivo de salida del método pero con
                //acentos
                ruta_traits_known=Constantes.TRAITS_KNOWN;
                ruta_relativa_salida=
                      "documentos/salidas/traits_known.xml";
                File file_traits_known=new File(ruta_traits_known);

                //Creamos el codificador para normalizar el XML
                Canonizador canonizador=new Canonizador();

                try {
                      //Rellenamos el archivo con acentos
                      canonizador.descanoniza(file_traits_known_cod,
                            file_traits_known);

                } catch (IOException e1) {

                      e1.printStackTrace();
                }

          }
          else
                System.out.println("El servicio no ha devuelto nada");

          return ruta_relativa_salida;

     }


     /**
      * Método que recibe todos los datos necesarios para invocar al
método get_profile de la interfaz ProfileAccess del Servicio Web.
      *
      * @param id String que contiene el identificador de una persona.
      * @param ruta_traits_requested Ruta del fichero de entrada
traits_requested.xml
      * @return Ruta relativa del fichero de salida profile.xml
      */
     public String get_profile_cli(String id , String
ruta_traits_requested){

          String ruta_profile=null;
          String ruta_profile_cod=null;
          String ruta_relativa_salida=null;

          //PROBAMOS EL MÉTODO get_profile

          OMElement traits_requested_cod=null;
          String contenido_traits_requested_cod = null;
          String ruta_traits_requested_cod=null;

          //Creamos el archivo de entrada que se supone que recibe el
          //programa con acentos
          File file_traits_requested = new File(ruta_traits_requested);
```

```
            //Creamos el codificador para normalizar el XML
            Canonizador canonizador=new Canonizador();


            //Creamos el archivo de entrada que recibe el método sin
            //acentos
            ruta_traits_requested_cod=Constantes.TRAITS_REQUESTED_COD;
            File file_traits_requested_cod=
                  new File(ruta_traits_requested_cod);

            //Rellenamos el archivo sin acentos
            try {

                  canonizador.canoniza(file_traits_requested,
                        file_traits_requested_cod);

            } catch (IOException e1) {

                  e1.printStackTrace();
            }

            traits_requested_cod=
                  Auxiliar.get_OMElement(ruta_traits_requested_cod);

            try {

                  contenido_traits_requested_cod =
                        traits_requested_cod.toStringWithConsume();

            } catch (XMLStreamException e) {

                  e.printStackTrace();
            }

            System.out.println("El contenido del fichero
traits_requested_cod.xml es el
siguiente:\n"+contenido_traits_requested_cod);

            //Preparamos el OMElement que indicará al servicio el ID y los
            //atributos
            OMElement peticion= fac.createOMElement("get_profile", omNs);
            OMElement value1 = fac.createOMElement("id", omNs);
            value1.addChild(fac.createOMText(value1, id));
            peticion.addChild(value1);

            OMElement value2= fac.createOMElement("traits_requested",
                  omNs);
            value2.addChild(fac.createOMText(value2,
                  contenido_traits_requested_cod));
            System.out.println("El contenido del fichero
traits_requested_cod.xml es el siguiente:
\n"+contenido_traits_requested_cod);
            peticion.addChild(value2);

            Options options = new Options();
            options.setTo(targetEPR);
```

```
            options.setTransportInProtocol(Constants.TRANSPORT_HTTP);


            ServiceClient sender;

            //OMELement que recibiremos del servicio y del que obtendremos
            //el fichero traits_known.xml
            OMElement resultado=null;

            try {

                  sender = new ServiceClient();

                  sender.setOptions(options);

                System.out.println("Invocamos al servicio ProfileAccess:
get_profile");

                  resultado =sender.sendReceive(peticion);


            } catch (AxisFault e) {

                  System.out.println("Error al crear el ServiceClient");
                  e.printStackTrace();

            }

            if (resultado!=null){

                  //RECUPERAMOS EL FICHERO profile_cod.xml QUE NOS
                  //DECUELVE EL SERVICIO

                  OMElement profile= resultado.getFirstElement();
                  String contenido_profile_cod= profile.getText();

                  System.out.println("El contenido del fichero
profile_cod.xml es el siguiente:\n"+contenido_profile_cod);

                  // CREAMOS EL FICHERO profile_cod.xml (sin acentos)
                  ruta_profile_cod=Constantes.PROFILE_COD;
                  Auxiliar.crea_xml(contenido_profile_cod,
                        ruta_profile_cod);


                  //PREPARAMOS profile.xml (con acentos)

                  //Creamos el archivo de salida que se supone que devuelve
                  //el método sin acentos
                  File file_profile_cod = new File(ruta_profile_cod);


                  //Creamos el archivo de salida del método pero con
                  //acentos
                  ruta_profile=Constantes.PROFILE;
                  ruta_relativa_salida="documentos/salidas/profile.xml";
                  File file_profile=new File(ruta_profile);
```

```
                try {
                      //Rellenamos el archivo con acentos
                      canonizador.descanoniza(file_profile_cod,
                            file_profile);

                } catch (IOException e1) {

                      e1.printStackTrace();
                }

          }
          else
                System.out.println("El servicio no ha devuelto nada");

          return ruta_relativa_salida;


    }

    /**
     * Método que recibe todos los datos necesarios para invocar al
método get_profile_list de la interfaz ProfileAccess del Servicio Web.
     *
     * @param ids Tabla de Strings con el identificador de una persona o
varias personas.
     * @param ruta_traits_requested Ruta del fichero de entrada
traits_requested.xml
     * @return Ruta relativa del fichero de salida taggedProfileSeq.xml
     */
    public String get_profile_list_cli(String ids [],String
ruta_traits_requested){

          OMElement traits_requested_cod=null;
          String contenido_traits_requested_cod = null;
          String ruta_traits_requested_cod=null;
          String ruta_taggedProfileSeq=null;
          String ruta_taggedProfileSeq_cod=null;
          String ruta_relativa_salida=null;

          //Creamos el archivo de entrada que se supone que recibe el
          //programa con tildes
          File file_traits_requested = new File(ruta_traits_requested);

          //Creamos el codificador para normalizar el XML
          Canonizador canonizador=new Canonizador();


          //Creamos el archivo de entrada que recibe el método sin
          //acentos
          ruta_traits_requested_cod=Constantes.TRAITS_REQUESTED_COD;
          File file_traits_requested_cod=
                new File(ruta_traits_requested_cod);

          //Rellenamos el archivo sin acentos
          try {
                canonizador.canoniza(file_traits_requested,
                      file_traits_requested_cod);
```

```
            } catch (IOException e1) {

                  e1.printStackTrace();
            }


            traits_requested_cod=
                  Auxiliar.get_OMElement(ruta_traits_requested_cod);

            try {

                  contenido_traits_requested_cod =
                        traits_requested_cod.toStringWithConsume();

            } catch (XMLStreamException e) {

                  e.printStackTrace();
            }

            System.out.println("El contenido del fichero
traits_requested_cod.xml es el
siguiente:\n"+contenido_traits_requested_cod);

            int i=ids.length;
            System.out.println("El número de ids solicitados es:"+i);
            int indice=0;

            //Preparamos el OMElement que indicará al servicio los ids y
            //el contenido de traits_requested.xml

            OMElement entrada=
                  fac.createOMElement("get_profile_list", omNs);
            OMElement value1 = fac.createOMElement("num_ids", omNs);
            String num_ids=""+i; // lo hacemos String
            value1.addChild(fac.createOMText(value1, num_ids));
            entrada.addChild(value1);


            //ANIDAMOS EN EL OMELEMENT entrada TODOS LOS IDS SOLICITADOS
            OMElement value2=null;

            while(i!=0)
            {
                  value2= fac.createOMElement("id", omNs);
                  System.out.println("id["+indice+"] vale: "+ids[indice]);
                  value2.addChild(fac.createOMText(value2, ids[indice]));
                  entrada.addChild(value2);
                  i=i-1;
                  indice=indice+1;
            }

            OMElement value3=
                  fac.createOMElement("traits_requested", omNs);
            value3.addChild(fac.createOMText(value3,
                  contenido_traits_requested_cod));
            entrada.addChild(value3);
```

```
            //Preparamos la llamada al servicio
            Options options = new Options();
            options.setTo(targetEPR);
            options.setTransportInProtocol(Constants.TRANSPORT_HTTP);


            ServiceClient sender;

            //OMELement que recibiremos del servicio y del que obtendremos
            //el fichero traits_known.xml
            OMElement resultado=null;

            try {

                  sender = new ServiceClient();

                  sender.setOptions(options);

                  System.out.println("Invocamos al servicio ProfileAccess:
get_profile_list");

                  resultado =sender.sendReceive(entrada);

            } catch (AxisFault e) {

                  System.out.println("Error al crear el ServiceClient");
                  e.printStackTrace();

            }

            if (resultado!=null){

                  //RECUPERAMOS EL FICHERO taggedProfileSeq_cod.xml QUE
                  //NOS DECUELVE EL SERVICIO

                  OMElement taggedProfileSeq_cod=
                        resultado.getFirstElement();
                  String contenido_taggedProfileSeq_cod=
                        taggedProfileSeq_cod.getText();

                  System.out.println("El contenido del fichero
taggedProfileSeq_cod.xml es el
siguiente:\n"+contenido_taggedProfileSeq_cod);

                  //CREAMOS EL FICHERO taggedProfileSeq_cod.xml
                  ruta_taggedProfileSeq_cod=
                        Constantes.TAGGEDPROFILESEQ_COD;
                  Auxiliar.crea_xml(contenido_taggedProfileSeq_cod,
                        ruta_taggedProfileSeq_cod);



                  //Creamos el archivo de salida que se supone que devuelve
                  //el método sin acentos
                  File file_taggedProfileSeq_cod =
                        new File(ruta_taggedProfileSeq_cod);
```

```
                  //PREPARAMOS taggedProfileSeq.xml (con acentos)
                  //Creamos el archivo de salida del método pero con
                  //acentos
                  ruta_taggedProfileSeq=Constantes.TAGGEDPROFILESEQ;
                  ruta_relativa_salida=
                        "documentos/salidas/taggedProfileSeq.xml";
                  File file_taggedProfileSeq=
                        new File(ruta_taggedProfileSeq);


                  try {
                        //Rellenamos el archivo con acentos

                        canonizador.descanoniza(file_taggedProfileSeq_cod,
                              file_taggedProfileSeq);
                  } catch (IOException e1) {

                        e1.printStackTrace();
                  }


            }
            else
                  System.out.println("El servicio no ha devuelto nada");


            return ruta_relativa_salida;

      }

}
```

Auxiliar.java

```
package Client;

import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.impl.builder.StAXOMBuilder;


/**
 * Clase donde se implementan los métodos crea_xml y get_OMELement.
 *
 * @author Pilar Mª Juan Montilla
```

```
 *
 */
public class Auxiliar {

	/**
	 * Método que crea un fichero XML a partir de su contenido y su ruta
absoluta.
	 *
	 * @param contenido_xml String con el contenido del fichero XML a
crear.
	 * @param ruta_fichero String con la ruta del fichero XML a crear.
	 *
	 */
	public static void crea_xml (String contenido_xml, String
ruta_fichero){

		try {

			FileWriter file_xml = new FileWriter (ruta_fichero);
			BufferedWriter writer_xml= new BufferedWriter(file_xml);

			writer_xml.write("<?xml version=\"1.0\" encoding=\"UTF-
8\"?>\n");
			writer_xml.write(contenido_xml);
			writer_xml.flush();

		} catch (IOException e) {
			System.out.println("Se ha producido el siguiente error en
el método crea_xml:\n");
			e.printStackTrace();


		}
	}


	/**
	 * Método que obtiene el OMElement correspondiente a un fichero XML.
	 *
	 * @param ruta_fichero_xml String que contiene la ruta del fichero
XML en cuestión.
	 * @return El OMELement de salida
	 */
	public static OMElement get_OMElement (String ruta_fichero_xml){

		OMElement salida=null;
		try {

			//create the parser
			XMLStreamReader parser =
			XMLInputFactory.newInstance().createXMLStreamReader(
			new FileInputStream(ruta_fichero_xml));
			StAXOMBuilder builder= new StAXOMBuilder(parser);
			//get the root element
			salida= builder.getDocumentElement();

		} catch (XMLStreamException e) {
```

```
                    System.out.println("Se ha producido el siguiente error en
el método get_OMElement:\n");
                    e.printStackTrace();

             } catch (FileNotFoundException e) {

                    System.out.println("El fichero XML no existe:\n");
                    e.printStackTrace();
             }

             return salida;

      }

}
```

Contantes.java

```
package Client;

/**
 * Clase donde se definen las constantes con las rutas de los documentos
XML que se crean en el cliente.
 *
 * @author Pilar Mª Juan Montilla
 *
 */
public class Constantes {

      /**
       * Ruta donde se crea el fichero profile_selector.xml
       */
      public static final String PROFILE_SELECTOR="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/entradas/profile_selector.xml";

      /**
       * Ruta donde se crea el fichero profile_selector_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
       */
      public static final String PROFILE_SELECTOR_COD="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/intermedios/profile_selector_cod.xml"
;

      /**
       * Ruta donde se crea el fichero traits_requested.xml
       */
      public static final String TRAITS_REQUESTED="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/entradas/traits_requested.xml";

      /**
```

```
       * Ruta donde se crea el fichero traits_requested_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
       */
      public static final String TRAITS_REQUESTED_COD="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/intermedios/traits_requested_cod.xml"
;

      /**
       * Ruta donde se crea el fichero returned_sequence.xml
       */
      public static final String RETURNED_SEQUENCE="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/salidas/returned_sequence.xml";

      /**
       * Ruta donde se crea el fichero returned_sequence_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
       */
      public static final String RETURNED_SEQUENCE_COD="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/intermedios/returned_sequence_cod.xml
";

      /**
       * Ruta donde se crea el fichero returned_iterator.xml
       */
      public static final String RETURNED_ITERATOR="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/salidas/returned_iterator.xml";

      /**
       * Ruta donde se crea el fichero returned_iterator_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
       */
      public static final String RETURNED_ITERATOR_COD="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/intermedios/returned_iterator_cod.xml
";

      /**
       * Ruta donde se crea el fichero profile.xml
       */
      public static final String PROFILE="/usr/local/apache-tomcat-
cliente/webapps/PIDSClient/documentos/salidas/profile.xml";

      /**
       * Ruta donde se crea el fichero profile_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
       */
      public static final String PROFILE_COD="/usr/local/apache-tomcat-
cliente/webapps/PIDSClient/documentos/intermedios/profile_cod.xml";

      /**
       * Ruta donde se crea el fichero traits_known.xml
       */
      public static final String TRAITS_KNOWN="/usr/local/apache-tomcat-
```

```
cliente/webapps/PIDSClient/documentos/salidas/traits_known.xml";

      /**
       * Ruta donde se crea el fichero traits_known_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
       */
      public static final String TRAITS_KNOWN_COD="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/intermedios/traits_known_cod.xml";

      /**
       * Ruta donde se crea el fichero taggedProfileSeq.xml
       */
      public static final String TAGGEDPROFILESEQ="/usr/local/apache-
tomcat-cliente/webapps/PIDSClient/documentos/salidas/taggedProfileSeq.xml";

      /**
       * Ruta donde se crea el fichero taggedProfileSeq_cod.xml ("cod" de
codificado, es decir, sin acentos ni caracteres 'ñ')
       */
      public static final String TAGGEDPROFILESEQ_COD="/usr/local/apache-
tomcat-
cliente/webapps/PIDSClient/documentos/intermedios/taggedProfileSeq_cod.xml"
;

      /**
       * Ruta del mapa PIDs.xml, necesario para la transformación Java-XML
mediante el uso de Castor
       */
      public static final String PIDS="/usr/local/apache-tomcat-
cliente/webapps/PIDSClient/public_html/PIDs.xml";


}
```