4.- Servicio de identificación de personas de CORBAmed

4.1.-EL OMG, Object Management Group

El OMG, es una organización internacional sin ánimo de lucro, formada por unos 800 miembros, incluyendo distribuidores de sistemas informáticos, desarrolladores de software y usuarios.
Fundado en 1989, el OMG promueve la teoría y práctica del paradigma de orientación a objetos en el desarrollo del software, y tiene como misión la creación de estándares para la integración de sistemas por medio de la orientación a objetos. Además, los objetivos de la organización también incluyen el establecimiento de una línea de industria y especificaciones sobre el manejo de objetos, que provean de un sistema común para el desarrollo de aplicaciones.
Son metas primarias la reutilización, portabilidad e interoperatividad del software orientado a objetos dentro de entornos heterogéneos y distribuidos.
El ceñirse a estas especificaciones hará posible desarrollar un entorno de aplicaciones heterogéneas portable a la mayoría de plataformas de hardware y sistemas operativos.

4.2.-CORBA y CORBAmed

CORBA, Common Object Request Broker Arquitecture (arquitectura común de intermediarios en peticiones a objetos), es un estándar desarrollado por el OMG en respuesta a la necesidad de interoperatividad entre la gran gama de productos hardware y software disponibles hoy día. En pocas palabras, CORBA permite la comunicación entre aplicaciones con independencia de su localización o de quién las creó.
CORBA ha logrado parte de su éxito gracias a la clara separación entre la interfaz y el objeto. La interfaz define qué servicios ofrece el objeto, cómo invocarlos y su implementación. Se define por medio de un lenguaje propio conocido como 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.
Como paso posterior al desarrollo de CORBA, el OMG ha puesto en marcha una serie de grupos de trabajo con el propósito de adaptar este estándar a un conjunto de sectores, entre los cuales se encuentra el sanitario. CORBAmed es el grupo de trabajo que el OMG ha creado para la adaptación de CORBA al 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 entre servicios y funciones sanitarias.
- Mejora la calidad de la atención sanitaria y reduce costes gracias al uso de tecnología CORBA.

4.3.-Necesidad del servicio de identificación de personas (PIDS)

Una persona a lo largo de su vida, puede llegar a recibir cuidados médicos suministrados por docenas o cientos de diferentes profesionales sanitarios, la mayoría de los cuales asignan 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. Ésto dificulta el acceso al historial clínico completo del paciente.
El problema del manejo de los identificadores se agrava por el hecho de que cada organización trata sus datos de forma distinta.No obstante, la 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 puede almacenarse y usarse posteriormente con fines identificativos.
El servicio de identificación de personas de CORBAmed (PIDS), maneja identificadores y gestiona información demográfica para cumplir las necesidades exigidas en el ámbito sanitario. El servicio ha sido diseñado para:

- 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 federados.

4.4.-Modelo de referencia del PIDS

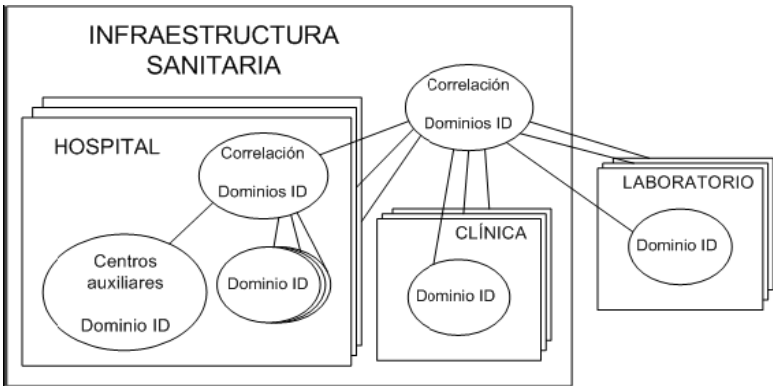

Figura 4.1: Modelo de referencia del PIDS

Comúnmente, un hospital maneja identificadores de persona pertenecientes a su propio dominio de identificación, y trabaja conjuntamente 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 de identificación de manera que se conviertan en uno solo.
De hecho, una buena infraestructura sanitaria contará con múltiples hospitales, laboratorios, clínicas... cada uno con su propio dominio de identificación, por lo que una correcta gestión de dichos dominios es fundamental.

4.5.-Modelo de identificación del PIDS

Se puede observar a continuación el modelo de identificación del PIDS:

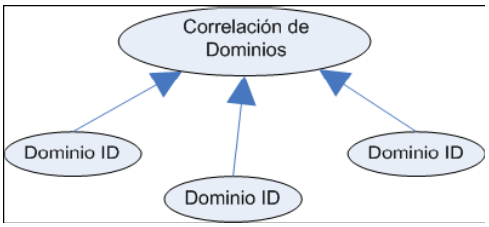

Figura 4.2: Modelo de identificación del PIDS

El Dominio ID, o dominio de identificación, es el bloque básico del modelo PIDS. Un dominio de identificación mantiene un único identificador (ID) para cada persona representada dentro del dominio. 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 sería imposible distinguirlas.
La unión del identificador personal y el identificador del dominio, crea un único identificador para la persona.
Las especificaciones del PIDS detallan varias interfaces. Dos de las principales dentro de un dominio ID son la interfaz **IdentifyPerson** y **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).

4.6.-Diagrama de herencia

El PIDS se estructura como un componente con múltiples interfaces que pueden ser implementadas por cualquier instancia del servicio. Cada interfaz representa un trozo de funcionalidad y es opcional, por lo que cada implementación del PIDS implementa sólo las interfaces que necesita.
Como se aprecia en la siguiente figura, todas las interfaces heredan de IdentificationComponent. La funcionalidad IdentificationComponent, encapsula una tabla lógica con características de personas (atributos) emparejadas con un ID. De esta manera, partiendo de un ID se pueden conocer todas las características disponibles de una persona. Un IdentificationComponent, tiene un número opcional de interfaces que puede implementar.
A continuación se detallan los interfaces más relevantes para el desarrollo de este proyecto:

- IdentifyPerson.
- Parte de la interfaz ProfileAccess (métodos get_traits_known, get_profile y get_profile_list).

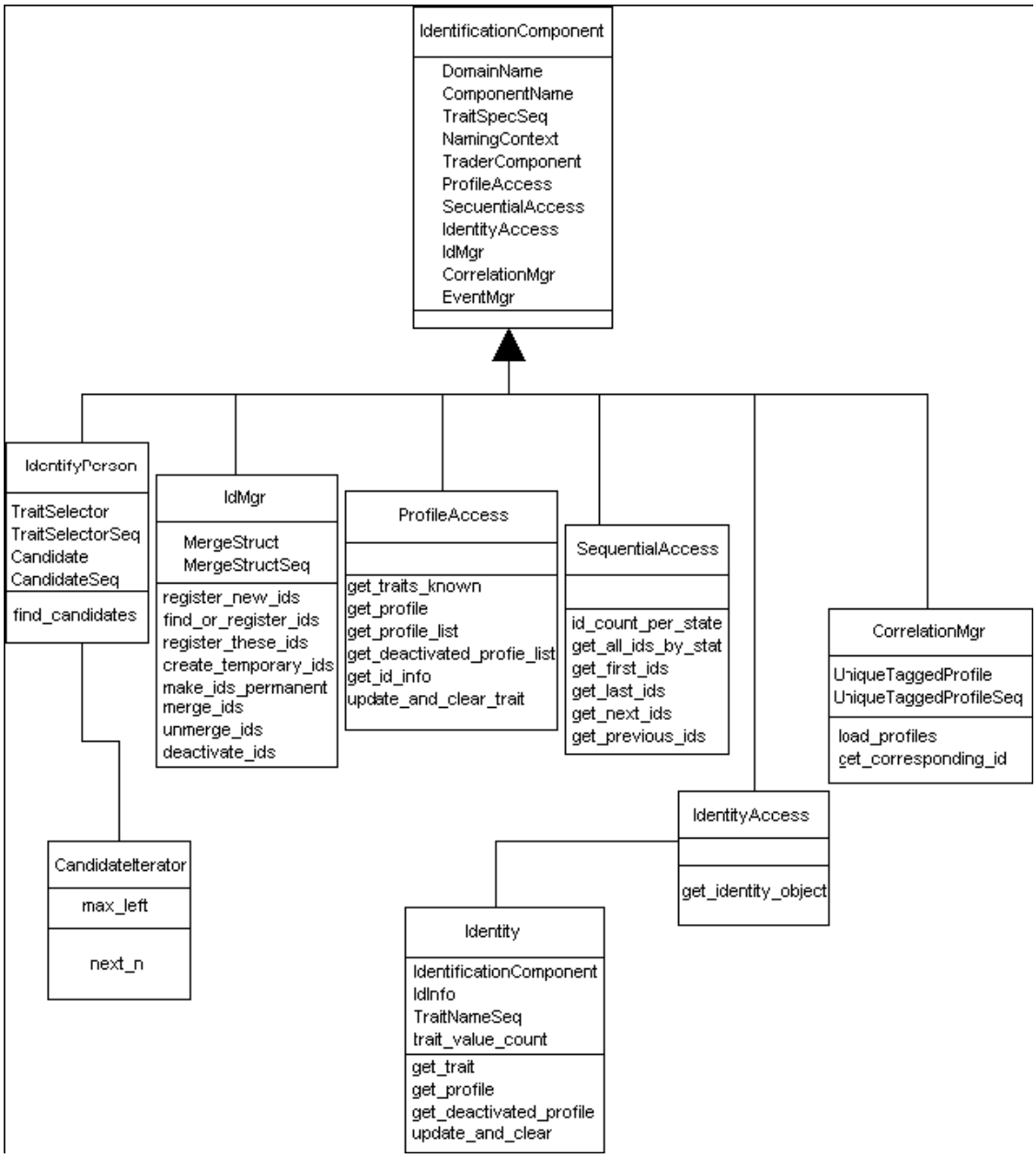

Figura 4.3: Diagrama de herencia del PIDS

4.7.- Interfaz IdentifyPerson

En ella se encuentran los siguientes componentes.

4.7.1.- TraitSelector y TraitSelectorSeq

TraitSelector es un parámetro de información que el cliente aporta al servicio para identificar a la persona o personas que 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;
};
```

También hay un campo weight (peso), que 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 componentes TraitSelector.

4.7.2.- 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 como de bien esa persona se ajusta 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 Candidates.
Por su parte, un Profile se define como:

```
        Typedef TraitSeq Profile;
```

Siendo TraitSeq una secuencia de componentes Trait.

4.7.3.- SpecifiedTraits

Es una secuencia de TraitSpec, que son nombres de atributos para los que el servicio debe encontrar su correspondiente valor.

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

4.7.4.- 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 (profile_selector), un cliente puede pedirle al servicio 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.

4.8.-Interfaz ProfileAccess

En ella se encuentran los siguientes componentes.

4.8.1.- TaggedProfile y TaggedProfileSeq

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

Estructura que indica un perfil para un ID. Un TaggedProfileSeq es una secuencia de TaggedProfile.

4.8.2.- ProfileUpdate y ProfileUpdateSeq

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

Estructura que indica para un ID qué atributos deben borrarse (del_list) y cuales modificarse (modify_list). ProfileUpdateSeq es una secuencia de ProfileUpdate.

4.8.3.- 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.

4.8.4.- 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 a través de traits_requested (secuencia de nombres de atributos), mediante un Profile (secuencia de Traits).

4.8.5.- 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 mas 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.

4.8.6.- 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.

4.8.7.- Método update_and_clear_traits

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

Se usa para modificar el perfil de un ID. La excepción MultipleTraits es similar a DuplicateTraits pero si aplica a operaciones en las que intervienes secuencias de IDs o de perfiles.

4.8.8.- Método get id info

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

Devuelve el estado de cada ID introducido.