Capítulo 6: Resultados

6.1 Metodología del modelo de información

6.1.1 Introducción

Uno de los mayores desafíos de la utilización de las tecnologías de la información dentro del ámbito sanitario consiste en proveer al personal sanitario, la información clínica necesaria que ayude a mejorar la calidad de atención sanitaria a los pacientes. La normalización de la historia clínica de un paciente supone un importante paso para alcanzar este objetivo, permitiendo una distribución más eficiente de la información entre instituciones y profesionales.

La tarea de integrar la información clínica de un paciente puede ir desde una simple distribución de datos entre dos fuentes de datos perteneciente a una misma institución, hasta la integración de toda la información de salud de un paciente que se encuentra dispersa en múltiples sistemas.

Dentro de este contexto, la norma del CEN/ISO 13606 supone un nuevo enfoque en el tratamiento de los datos, por un lado a través de una separación de puntos de vista denominado ODP (Procesamiento Distribuido Abierto), y por otro una separación de información y conocimiento (Modelo Dual). Esto permite el diseño de sistemas a partir de un modelo de referencia, que definen conceptos no volátiles, y por un modelo de arquetipos, que no son más que metadatos que especifican por medio de restricciones en el modelo de referencia, las características concretas de cada una de las estructuras de datos, extendiendo o especializando un determinado componente de información descrito en el modelo de referencia.

La visión de la norma CEN/ISO 13606 es la de habilitar sistemas clínicos heterogéneos y especializados para intercambiar todo o partes del HCE de un paciente de forma normalizada, de tal forma que puedan representarse los valores de datos y la organización contextual de la información. Un objetivo complementario ha sido conciliar la naturaleza desarrollada del conocimiento médico y la diversidad inherente a la práctica clínica. Al disponer de esta clara separación entre información y conocimiento, un sistema basado en el modelo dual es capaz de evolucionar y adaptarse de manera sencilla y automática ante los cambios producidos en las definiciones de conceptos clínicos.

La norma CEN/ISO 13606 está desarrollada bajo un conjunto de tipos de datos que hasta hace poco estaban estipulados en la norma TS14796 [60]. Sin embargo, debido a diversos problemas asociados a esta especificación, entre ellos la no homogeneidad con los tipos de datos del HL7, se ha anunciado la sustitución de esta especificación por una nueva: la norma ISO/DIS 21090 [61]. Asimismo, esta norma está basada en la ISO/IEC 11404 [62], en la que se especifican los tipos de datos más básicos.

El modelo de referencia de la norma CEN/ISO 13606 viene compuesto por una serie de entidades que están anidadas unas dentro de otras, y cuyos datos almacenados permanecen estáticos, es decir, no cambian con el tiempo. Al contrario que la información, el conocimiento sí puede variar o modificarse con el paso del tiempo, y además presenta un significado entendible para el profesional pertinente. Este conocimiento se representa mediante arquetipos según la norma CEN/ISO13606-2. Este conocimiento es el que, por diversas razones estrictamente médicas, puede llegar a variar con el paso del tiempo.

En la implementación realizada, para el intercambio de información entre diferentes

entidades de comunicación DDS, se han desarrollado diferentes topics basados en estándares sanitarios. Inicialmente se comenzó empleando el modelo de referencia y de arquetipos descrito en la norma CEN/ISO 13606, pero debido la gran variabilidad de campos y componentes, se trata de una tarea extensa e inabordable por cuestiones de limitación. Se optó por tanto, en adaptar la norma ISO/DIS 21090, que ha sido liberada recientemente, como base para la composición de topics. A pesar de esto, la norma CEN/ISO 13606 está basada en la ISO/DIS 21090, así que el trabajo realizado es extensible para utilizar el modelo de referencia y el de arquetipos en un futuro.

Fruto del trabajo previo desarrollado con la norma CEN/ISO 13606, se adaptó en primer lugar al lenguaje IDL la norma CEN/TS 14796, que especifica los tipos de datos más básicos que componen las entidades del modelo de referencia. Las dificultades encontradas en su implementación son extrapolables a las existentes en la ISO/DIS 21090, por lo que la posterior adaptación de esta norma a IDL fue prácticamente directa.

En capítulos sucesivos se comentará por tanto los contratiempos que entraña esta adaptación a IDL, comenzando con la CEN/TS 14796:2005 y posteriormente con la actualización del 2007 de esta misma norma. En el apartado 6.1.3 se describirá de forma breve la conformación de la ISO/DIS 21090, y por último, se especificarán los topics que finalmente son empleados en la implementación.

6.1.2 Adaptación de la norma CEN/TS 14796:2005 al lenguaje IDL

El primer paso consiste en la adaptación de los tipos de datos básicos recogidos en la norma CEN/TS 14796, y que constituyen los componentes del modelo de referencia, a los tipos de datos empleados en la definición de topics mediante IDL.

Las clases que forman el modelo abstracto de la norma CEN/TS 14796 están agrupadas por comodidad en paquetes, tal y como se muestra en la Figura 6.1:

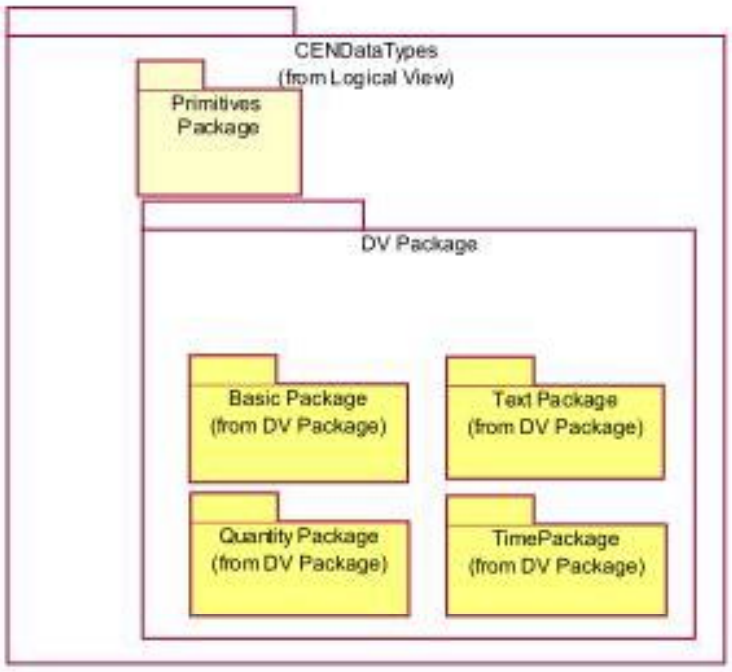

Figura 6.1 Paquete CENDataTypes

En la implementación de los tipos de datos contenidos en esta norma, se ha tomado como referencia la plantilla XML que forma parte del editor de arquetipos LinkEHRArchetype Editor [71], y que incluye entre sus paquetes una descripción formal de los tipos básicos de datos de la norma CEN/TS 14796:2005. Este editor fue desarrollado por el Grupo de Informática Biomédica de la UPV (Universidad Politécnica de Valencia) y ha sido liberado como software libre.

Para la especificación de estos tipos básicos de información que se van a manejar, es necesario una adaptación a los que se definen en el lenguaje de definición de interfaces o IDL, para constituir la estructura de los topics con los que va tratar el sistema. Sin embargo, debido a las actuales limitaciones de la especificación DDS, no existe a día de hoy una alternativa estandarizada para definir tipos de datos complejos, ni topics dinámicos cuya estructura pueda concretarse en tiempo de ejecución. Para afrontar estas adversidades, la OMG se encuentra trabajando en un concepto nuevo denominado X-Topic (Extensible and Dynamics Topic) [63], para introducirlo en la próxima revisión DDS.

El compilador utilizado para la generación de código de los interfaces especializados DDS/DCPS (*TypeSupport*, *DataReader* y *DataWriter*) es el *idlpp*, que proporciona OpenSplice DDS. Soporta dos modos de funcionamiento:

- Modo *standalone*, donde la aplicación solo hace uso del middleware de OpenSplice DDS
- *Modo ORB integrated*, para integrar DDS con CORBA utilizando el ORB OpenFusion TAO o JacORB (dependiendo del lenguaje de programación empleado).

Para un determinado lenguaje de programación especificado (C, C++ o Java), el Pre-procesador OpenSplice DDS IDL genera los interfaces para *TypeSupport*, *DataReader* y *DataWriter* a partir de las plantillas proporcionadas por OpenSplice.

El pre-procesador IDL acepta la gramática conforme a la especificación CORBA, pero ignora los elementos no relevantes en la definición de los tipos de datos para DDS.

La implementación realizada en IDL de esta norma está contenida dentro de los archivos:

- BasicPackage.idl
- TextPackage.idl
- QuantityPackage.idl
- TimePackage.idl
- DVPackage.idl
- PrimitivesPackage.idl
- CENDataTypes.idl

Si se quiere hacer uso de todos los paquetes, se importaría el archivo DVPackage.idl, o el CENDataTypes.idl, indistintamente, conteniendo la implementación completa de la norma CEN/TS 14796:2005.

Algunas características que conviene aclarar respecto a la implementación son las siguientes:

- Los nombres de las clases estructuradas definidas tienen sus nombres en mayúscula, mientras que sus atributos se muestran en minúscula (tal y como se describe en la norma).

- Cada uno de los archivos idl anteriores definen estructuras de datos conforme a un determinado paquete de la norma, utilizan tipos de datos concernientes a otros paquetes, y por tanto a otros archivos. Aunque el pre-procesador IDL de OpenSplice incorpora la directiva #include para la importación de otros archivos, las relaciones tan íntimas de los tipos de datos entre diferentes paquetes acarrean problemas de recursividad. Para resolver esto, se hace uso de las directivas del procesador #ifdef y #ifndef, que son heredadas del lenguaje C.
- Las clases abstractas no están implementadas porque no se instancian. Sus atributos pasan a heredarse a sus clases derivadas.

A continuación se pasan a describir cada uno de los paquetes y las contingencias surgidas a la hora de adaptarlos a la gramática IDL.

Tipos de datos primitivos

El paquete de tipos de datos primitivos está compuesto por tipos que son atómicos, es decir, aquellos que no están definidos como compuestos de otros tipos de datos. Estos tipos de datos atómicos se observan en la Figura 6.2.

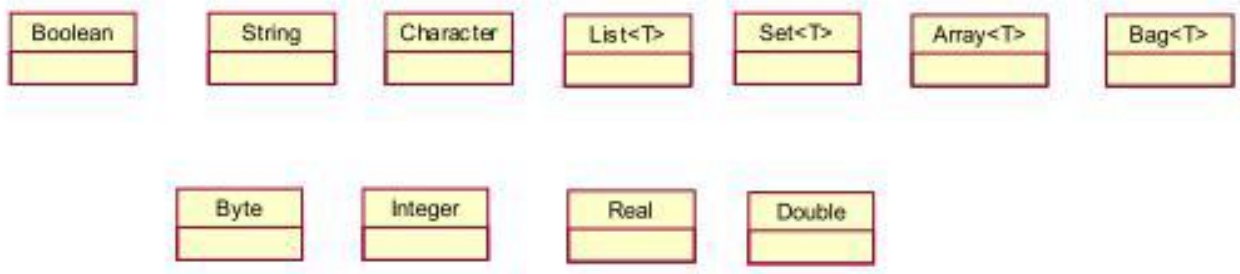

Figura 6.2 Tipos de datos primitivos

La conversión a IDL es inmediata a partir de esta tabla:

Tabla VI: Conversión entre tipos de datos básicos del CEN/TS 14796 el lenguaje IDL

CEN/TS 14796	DDS 1.2 (IDL-CORBA)
Boolean	boolean
String	string
Character	char
Byte	octet
Integer	long
Real	float
Double	double

El tipo byte viene representado por el tipo *octet* en IDL, suponiendo que un byte son 8 bits, aunque en los computadores antiguos no siempre ha existido esta equivalencia.

Los tipos *List<T>*, *Set<T>*, *Bag<T>*, *Array<T>*, donde T es un tipo de dato primitivo u otro

tipo de dato básico, se han definido como tipo unión, utilizando la partícula switch para definir el tipo de dato en tiempo de ejecución. Por ejemplo, en la Figura 6.3 se expone el código para el tipo de dato *List*<T>:

```
union List switch (T) {
      case Boolean_s:
            sequence<boolean> List_Boolean;
      case String_s:
            sequence<string> List_String;
      case Character_s:
            sequence<char> List_Character;
      case Byte_s:
            sequence<octet> List_Byte;
      case Integer_s:
            sequence<short> List_Integer;
      case Real_s:
            sequence<float> List_Real;
      case Double_s:
            sequence<double> List_Double;
};
```

Figura 6.3 Implementación del tipo *List*<T>

En donde la variable de tipo secuencia va a denominarse *List_x*, y *x* va a denotar el tipo de elemento que almacena la secuencia. A pesar de que todos los elementos han sido implementados como secuencias, hay que tener en cuenta las siguientes diferencias:

- Un conjunto (*Set*): los elementos aparecen sin un orden concreto.
- Una lista (*List*) es una secuencia ordenada de valores discretos.
- Una bolsa (*Bag*) es una colección desordenada de elementos, en que cada elemento puede estar contenido más de una vez en la bolsa.
- Un array (*Array*) es un contenedor físico de elementos indexados por un número.

Por su parte, T es un tipo enumerado formado por todos los tipos de datos primitivos y básicos que no son un conjunto. En la Figura 6.4 se presenta la implementación de este tipo.

```
enum T {
      Boolean_s,
      String_s,
      Character_s,
      List_s,
      Set_s,
      Array_s,
      Bag_s,
      Byte_s,
      Integer_s,
      Real_s,
      Double_s
};
```

Figura 6.4 Implementación del tipo enumerado T

Valor de los datos (DATA_VALUE o DV)

Es un tipo abstracto que define las propiedades básicas para cada tipo de valor de datos. Cada tipo estructurado es una especialización de este tipo de datos abstracto general.

En la gramática IDL de DDS no se pueden definir tipos abstractos de datos, por lo que no se

especifica el tipo DV en la implementación realizada, y su atributo nullFlavor de tipo CS, pasa a heredarse a sus clases derivadas.

Tipos de datos básicos

Los tipos de datos básicos son todas las especializaciones de *DATA_VALUE*, y utilizan los tipos de datos primitivos descritos en el apartado anterior. Las relaciones entre clases de muestra en la figura 6.5.

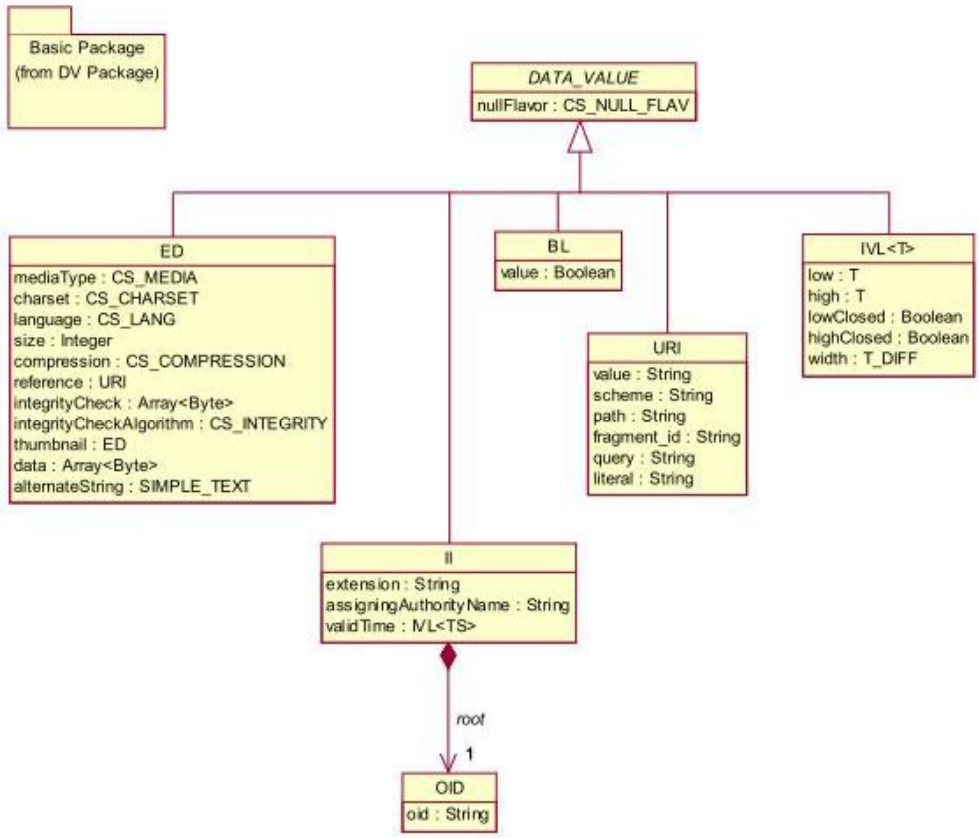

Figura 6.5 Paquete Basic Package

Donde las partículas *CS_NULL_FLAV*, *CS_MEDIA*, *CS_CHARSET*, *CS_LANG*, *CS_COMPRESSION* son especializaciones adicionales del tipo *CS* (del paquete *TextPackage*) y en su definición son tratadas como tal.

El array de bytes data, perteneciente al tipo ED, se ha convertido a una secuencia de octetos (*sequence<octet>*) para su adaptación a IDL, al igual que la variable *integrityCheck*.

Otro problema en la adaptación de tipos son los atributos que son instancias a la propia clase a la que pertenecen. Un ejemplo es el atributo *thumbnail* de tipo *ED*, perteneciente a la clase *ED*, cuya dependencia no puede ser resuelta por el precompilador *idlpp* de OpenSplice, y es advertida por el mismo como una excepción de coma flotante. La elección tomada para enfrentarse a este impedimento es la utilización del tipo de dato *sequence*. Una *sequence* o secuencia presenta dos características importantes: un tamaño máximo (definido en tiempo de compilación), y una longitud (determinada en tiempo de ejecución). Si el tamaño máximo no se especifica, no se establece uno

predeterminado para la secuencia, denominándose *unbounded*.

Teniendo en cuenta la problemática existente con la variable *thumbnail*, resulta posible declarar ésta como una secuencia cuyos componentes van a ser de tipo ED, sin asignar un valor máximo de elementos, y tomando únicamente el primer elemento de la secuencia para enlazar con la siguiente estructura de tipo ED, como si se tratase de una lista enlazada, tal y como se observa en la Figura 6.6.

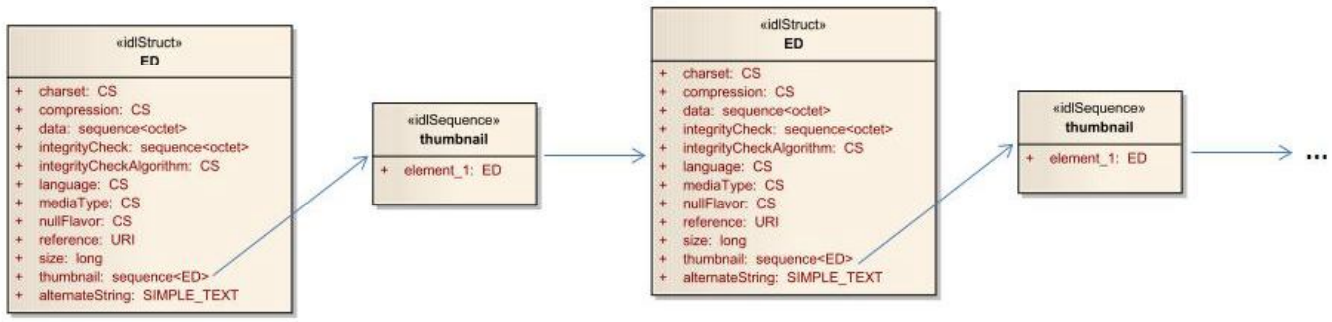

Figura 6.6 Variable thumbnail en tipo ED

En caso de que nos encontremos en el último elemento de esta lista, el valor que tomaría la longitud de la secuencia del atributo *thumbnail* sería 0.

Por último, para el tipo *IVL<T>*, donde *T* es un tipo de dato primitivo u otro tipo de dato básico, e IVL define un intervalo para un conjunto de valores, se emplean tres variables de tipo abstracto *QUANTITY* para especificar el límite inferior (*low*), otra para el límite superior (*high*), y otra para la diferencia entre ambos límites (*width*). Además, también existen dos atributos más de tipo *boolean* para indicar si el intervalo se encuentra cerrado o abierto.

Al igual que en el paquete *PrimitivesPackage*, *low_s*, *high_s* y *width_s* son de tipo unión, y el tipo *T* acompañado de la sentencia *switch* va a determinar el tipo de dato en tiempo de ejecución. En la Figura 6.7 se muestra la implementación para el tipo *low_s*:

```
union low_s switch (T) {
      case Boolean_s:
            boolean low_Boolean;
      case String_s:
            string low_String;
      case Character_s:
            char low_Character;
      case Byte_s:
            octet low_Byte;
      case Integer_s:
            short low_Integer;
      case Real_s:
            float low_Real;
      case Double_s:
            double low_Double;
};
```

Figura 6.7 Implementación del tipo low_s

Hay que tener en cuenta también que el tipo de dato *width* no siempre es del mismo tipo que

sus límites. Para diferencias entre magnitudes de escala (por ejemplo *Real PQ*), sí es el mismo. Para diferencia en magnitudes de otros tipos (por ejemplo *TS*), se emplea el tipo de datos DURATION.

Esta solución también se utiliza para la clase *CS*, que posee una instancia debido al atributo heredado de la clase abstracta *DATA_VALUE*.

Tipos de datos de texto y codificados

Este paquete es mostrado en la Figura 6.8, y proporciona clases que pueden utilizarse como tipos de valores para los siguientes propósitos:

1. *SIMPLE_TEXT* para una cadena son soporte para idioma y conjunto de caracteres.
2. *CODED_TEXT* para añadir un código a un *SIMPLE_TEXT*.
3. La clase abstracta *TEXT* puede especificarse cuando está permitido *SIMPLE_TEXT* o *CODED_TEXT*.
4. *CV* cuando sólo se requiere un código (opcionalmente con traducciones y calificadores)

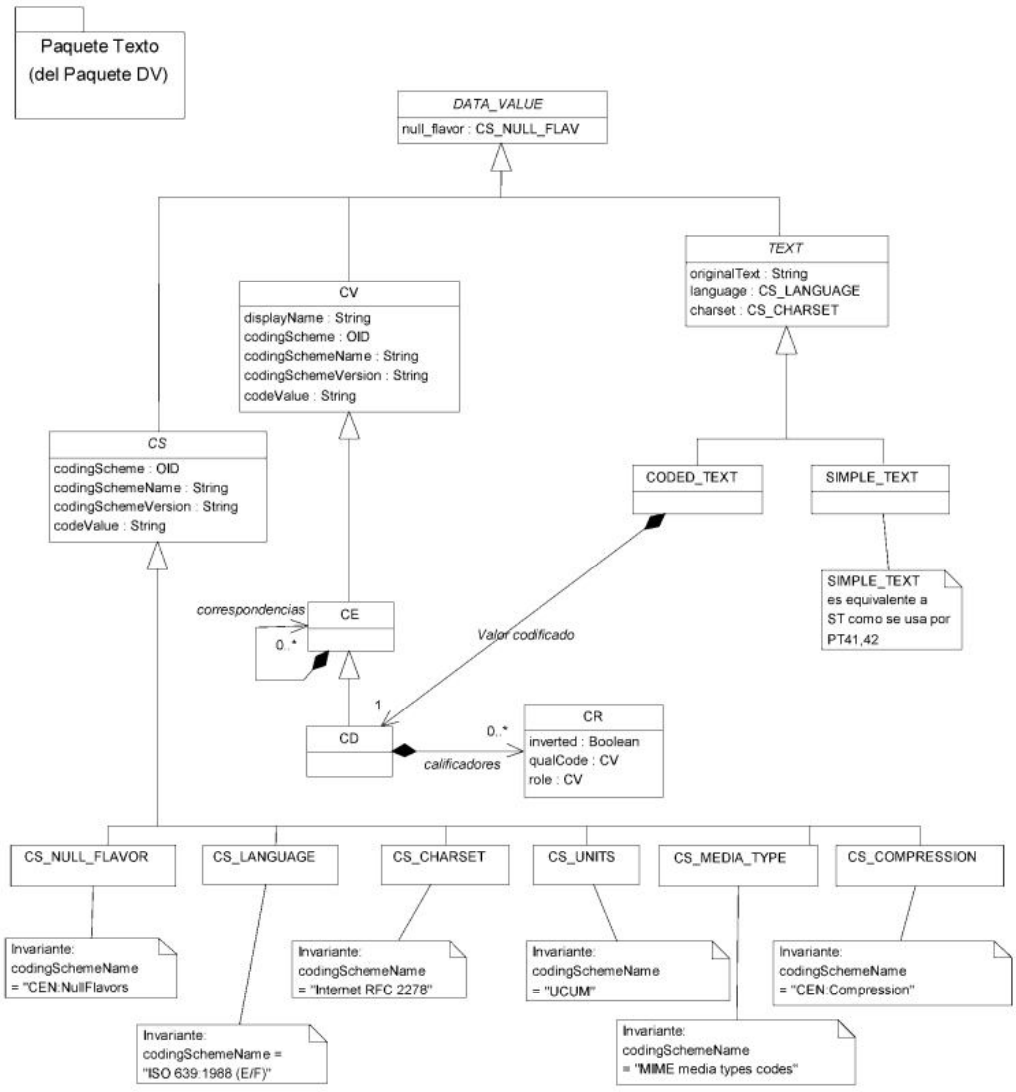

Figura 6.8 Paquete de texto

Los problemas surgidos a la hora de implementar el entramado de clases de este paquete y sus relaciones son similares al del apartado anterior, y se ha seguido estrictamente la plantilla XML que forma parte del editor de arquetipos *LinkEHRArchetype Editor*:

- Los tipos *CS_NULL_FLAV*, *CS_MEDIA*, *CS_CHARSET*, *CS_LANG*, *CS_COMPRESSION* son especializaciones del tipo *CS* definidas mediante restricciones que corren a cargo del desarrollador, por lo que no se han considerado en la implementación. Si se quiere utilizar alguno de estos tipos de variables, se recurrirá por tanto al tipo *CS*.
- El tipo *TEXT* es abstracto y por tanto no está implementado. Sus atributos se heredan a sus clases hijas *SIMPLE_TEXT* y *CODED_TEXT*. Éste último además posee un atributo adicional denominado *codedValue* de tipo *CD*.
- El tipo de dato *CD* posee un atributo adicional denominado *mappings*, que es un conjunto de variables de tipo *CD* y que ha sido adaptado a IDL como *sequence<CD>*.
- El tipo de dato *CE* posee un atributo adicional denominado *qualifiers*, que es una lista de variables de tipo *CR* y que ha sido adaptado a IDL como *sequence<CR>*.

Tipos de magnitud y tipos relacionados con el tiempo

La implementación de estos dos tipos de datos no reviste ninguna dificultad si se siguen las pautas tomadas en apartados anteriores. Dicha implementación está recogida en los archivos TimePackage.idl y QuantityPackage.idl**.**

6.1.3 Actualización de la norma CEN/TS 14796:2007 al lenguaje IDL

Aunque el editor de arquetipos *LinkEHRArchetype Editor* está basado en la norma CEN/TS 14796:2005, se incluye una actualización de la nueva revisión correspondiente al año 2007.

Las modificaciones respecto a la anterior residen en los paquetes *TextPackage* y *QuantityPackage*, y son mínimas. El resto permanece inalterable.

Las modificaciones son las siguientes:

- Las clases *RTO* y *ORD* pasan a ser derivadas de la clase abstracta *DATA_VALUE* dentro del paquete QuantityPackage.
- En el paquete TextPackage, la clase *CD* pasa a ser derivada de la clase abstracta *DATA_VALUE*, mientras que las clases *CE* y *CV* pasan a ser derivadas de la primera. *CS*, que antes era derivada de *DATA_VALUE*, pasa a ser derivada de *CV*

La implementación de esta nueva actualización está exenta de dificultad a partir de las medidas tomadas en el apartado anterior

6.1.4 Adaptación de la norma ISO/DES 21090 al lenguaje IDL

La norma ISO/DES 21909 tiene como objetivo proveer un conjunto de definiciones de tipos de datos para la representación y el intercambio de conceptos clínicos básicos que comúnmente son encontrados en entornos sanitarios. Suministra las definiciones en la terminología y notación de

UML, y especifica un XML basado en la representación de tipos de datos para su uso en el intercambio de información entre entidades. Su desarrollo está basado principalmente en la norma HL7 versión 3, en la ISO/IEC 11404, en la CEN/ISO 13606, y en otros trabajos de la ISO relacionados con tipos de datos sanitarios.

Se presenta como una norma muy extensa, por lo que no se ha implementado por completo y únicamente se ha concentrado la atención en aquellos paquetes que engloban tipos de datos utilizados en la composición de topics, así como otros paquetes necesarios que dependen de los primeros.

Los archivos implementados que recogen cada uno de los diferentes paquetes son los siguientes:

- ISO_1404.idl
- ISO_21090.idl
- ISO_21090_BasicDataTypes.idl
- ISO_21090_CodedDatatypes.idl
- ISO_21090_ContinuousSetDatatypes.idl
- ISO_21090_IdentificationandLocationDatatypes.idl
- ISO_21090_NameandAddressDatatypes.idl
- ISO_21090_QuantityDatatypes.idl
- ISO_21090_TextAndBinaryDataTypes.idl

El archivo ISO_21090 reúne la norma completa implementada, y es el que se importa a la hora de constituir los topics. Su implementación está recogida en la Figura 6.9.

```
#define __ISO_21090
#ifndef __ISO_1404
#include "ISO_1404.idl"
#endif

#include "ISO_21090_BasicDataTypes.idl"
#include "ISO_21090_CodedDatatypes.idl"
#include "ISO_21090_NameandAddressDatatypes.idl"
#include "ISO_21090_TextAndBinaryDataTypes.idl"
#include "ISO_21090_QuantityDatatypes.idl"
#include "ISO_21090_IdentificationandLocationDatatypes.idl"
#include "ISO_21090_ContinuousSetDatatypes.idl"
```

Figura 6.9 Archivos implementados para la norma ISO 21090

Los tipos de datos atómicos están descritos en la norma ISO/IEC 11404, y su estructura es análoga a la CEN/TS 14796, comentada con anterioridad, por lo que la metodología a seguir es similar en este caso. Igualmente sucede a la hora de implementar los diferentes paquetes de la ISO/DES 21090.

6.2. Servicio distribuido de control de glucosa en sangre

6.2.1 Introducción

Con el objetivo de que el presente trabajo no se presente como una memoria puramente teórica, se implementa un servicio distribuido de control de glucosa en sangre para pacientes diabéticos de tipo I.

Se ha planteado un escenario de monitorización y control de glucosa que puede estar corriendo en plataformas dispares que no se han considerado, al igual que tampoco se ha especificado una tecnología de red en cuestión. El equipo de pruebas ha sido únicamente un portátil, con varias aplicaciones que intercambian datos por medio de DDS.

Sin embargo, el escenario diseñado solo es una muestra sencilla del potencial de un middleware como DDS para el intercambio de datos en tiempo real entre sistemas heterogéneos que pueden estar ejecutándose en lugares distintos. En la Figura 6.10 se muestra varios entes de comunicación con la capacidad de generar o consumir información, y que tienen como punto de unión la “nube” DDS.

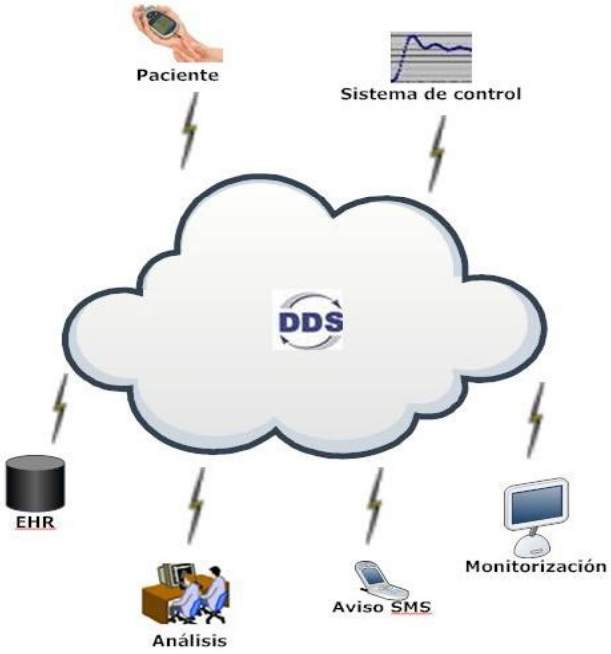

Figura 6.10 Escenario planteado

En la implementación realizada, el escenario se reduce, con 4 entes de comunicación que interactúan con 3 topics y 1 topic filtrador de contenido. Las relaciones entre estos elementos se ilustran en la Figura 6.11.

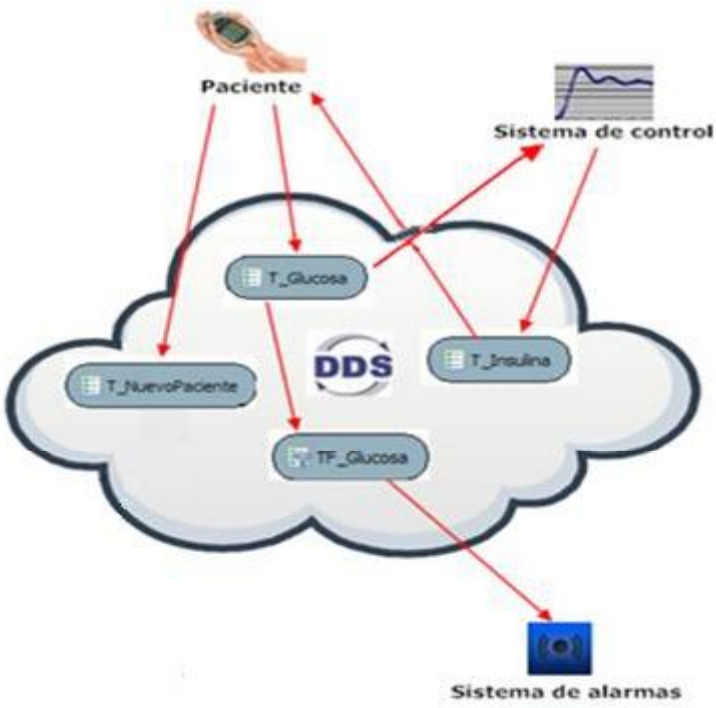

Figura 6.11 Escenario implementado

Las aplicaciones participantes en el dominio DDS son las siguientes:

- Paciente: se utiliza el modelo matemático de Bergman, que corre en Simulink, y es comentado brevemente en el apartado 6.2.2 de este capítulo. Básicamente, el modelo publica glucosa en el topic correspondiente, y está suscrito al topic de insulina del que toma los valores publicados por el servicio de control para completar el camino de retroalimentación.
- Sistema de control: está suscrito al topic de glucosa y se encarga de publicar insulina en función del controlador asociado al paciente.
- Sistema de alarmas: está suscrito al topic filtrado de contenido TF_Glucosa, y recibe muestras cuando los valores de glucosa para un paciente se encuentra por encima de 140mg/dL, o por debajo de 70mg/dL.

6.2.2 Modelo de Bergman

La diabetes es una enfermedad crónica para la cual no se tiene cura, y se caracteriza por un control inadecuado del nivel de glucosa en sangre por parte del páncreas, que segrega 2 hormonas para dicho control: insulina y glucagón. La insulina baja el nivel de glucosa en sangre, mientras el glucagón la sube.

Existen 2 tipos de diabetes: diabetes mellitus 1 y diabetes mellitus 2. En el tipo 1, llamado insulino-dependiente, el páncreas no produce insulina en absoluto. Para controlar la glucosa, el paciente sólo depende de una fuente externa de insulina, que es suministrada de forma exógena para un apropiado control de la glucosa. Los niveles de concentración de glucosa en sangre considerados normales oscilan entre 70-140 mg/dL, aunque varían en función de la persona y de la fuente de donde se toma la información. Si la insulina se inyecta en exceso, la concentración de glucosa en sangre decae rápidamente, produciendo un estado de hipoglucemia, y pudiendo provocar desmayos e incluso la muerte. En el lado opuesto se encuentra el estado de hiperglucemia, que se alcanza cuando el nivel de glucosa se encuentra por encima de 140 mg/dL, y cuyas consecuencias pueden

acarrear efectos muy desagradables a largo plazo, como la destrucción de órganos y patologías relacionadas. En primera instancia, y aunque las 2 son muy perjudiciales, se considera más dañino el estado de hipoglucemia, por lo que a la hora de evaluar un determinado algoritmo de control, debe penalizarse más cuando la glucosa decaiga por debajo de 70 mg/dL.

En este trabajo solo se considerarán pacientes de tipo 1, y para su estudio se utilizará el modelo mínimo de Bergman [64], que suele utilizarse para probar la bondad de diversos controladores, como ya realizan otros autores [65].

Las ecuaciones del modelo se recogen en la figura 6.12.

$$\frac{dG}{dt} = -P_1 G - X(G + G_b) + D(t)$$

$$\frac{dI}{dt} = -n(I + I_b) + \frac{U(t)}{V_I}$$

$$\frac{dX}{dt} = -P_2 X + P_3 I$$

$$D(t) = \frac{D_G A_G \, t \, e^{-t/t_{max,I}}}{V_G t^2_{max,G}}$$

Figura 6.12 Modelo de Bergman

Donde G es la concentración de glucosa en plasma sanguíneo dado en mmol/L, I es la concentración de insulina en plasma (mU/L), X es proporcional a la concentración de plasma en compartimiento remoto (min^{-1}), D(t) es la perturbación de comida en mmol/($Lmin^{-1}$), y U es la infusión de insulina exógena (mU/min).

Los parámetros para personas diabéticas toman los siguientes valores: P1=0, Gb=4.5 mmol/L, n=5/54 min^{-1}, VI=12 L, Ib=15 mU/L, P2=0.025 min^{-1}, P3=1.3e-5 L/m$Umin^2$, A_g = 0.8 (adimensional), V_g = 13.79 L, $t_{max,G}$ = 40min, $t_{max,I}$ = 55min.

6.2.3 Herramientas utilizadas y configuraciones

Para el desarrollo de la implementación se han empleado diferentes herramientas de trabajo. A continuación se expone cada una de ellas, categorizada en grupos, así como las configuraciones/modificaciones realizadas en las mismas y las dificultades entrañadas durante su manejo.

- Herramientas de simulación

  Para la simulación del modelo matemático de Bergman, así como la ejecución de los diversos controladores, se ha empleado la herramienta matemática Matlab, y Simulink como entorno visual a más alto nivel para la interacción entre sistemas.

  La conexión entre los servicios desarrollados en Java y la plataforma Matlab, se emplea

mediante el API matlabcontrol [66], que permite realizar llamadas remotas de Matlab desde Java. Para ello se crea una conexión proxy a nivel local, y a continuación se hace uso de los métodos *eval* o *returningEval*, dependiendo de si la función o comando llamado va a retornar algún valor o no.

Evidentemente, para que el API matlabcontrol funcione correctamente, Matlab debe estar instalado en el equipo. Actualmente el API matlabcontrol es compatible con los sistemas operativos Windows y Linux, con todas las versiones de Java a partir de la 1.6, y con las de Matlab a partir de la versión 2007.

Como se vio en capítulos anteriores, la comunicación entre Matlab y DDS no es directa, y es necesaria la intermediación de una base de datos que permita el trasvase de datos entre ambas plataformas. Por un lado, Matlab emplea el DatabaseToolbox [67] para insertar y extraer contenido de la base de datos mediante instrucciones SQL que son llamadas desde una función Matlab. Esta gestión de datos es realizada sobre una serie de tablas que deben de estar construidas previamente en la base de datos, y que se detallarán posteriormente en el manual del programador.

Otro factor importante a tener en cuenta en la simulación del modelo de Bergman en Simulink es la generación de datos en tiempo real. En este caso, Simulink no se encuentra preparado para abordar este contratiempo, y el tiempo de simulación real va a depender de diversos factores, como el tiempo de muestreo, el método de integración empleado, la complejidad del algoritmo implementado o el grado del controlador.

Por tanto, para simular un comportamiento de tiempo real, Matlab provee el bloque RTsyncBlockset [68] (Figura 6.13), que una vez ubicado en nuestro modelo, va a permitir que el tiempo de simulación configurado en Simulink sea estrictamente real.

Figura 6.13 RTsyncBlockset

Para nuestro caso, teóricamente el tiempo de simulación es alrededor de un día (1200 minutos). Este tiempo es suficientemente extenso como para controlar la evolución de la concentración de glucosa en sangre para un determinado paciente a lo largo del tiempo y con perturbaciones de comida cada 5 horas aproximadamente.

Debido a la inviabilidad de llevar a cabo una simulación de 24 horas, en el modelo de Simulink se ha optado por convertir los minutos en segundos, para agilizar la simulación del modelo. De esta forma, con el bloque de tiempo real RTsync, la simulación empleará 1200 segundos reales, cuando generalmente este proceso demoraría unos segundos, sin dicho bloque.

Gracias a este procedimiento va a ser posible que el tiempo entre muestra y muestra sea adecuado para que el modelo publique los datos de la concentración de glucosa en un topic en DDS, se lleve a cabo la adquisición y gestión de esta información por parte de la aplicación que gestiona el servicio de control de glucosa, y por último, se suscriba al topic de insulina que va a permitir la retroalimentación del modelo.

Por último, comentar que el tiempo de muestreo va a ser variable, y depende en gran medida

del modelo empleado. En este caso, Simulink necesita alrededor de 2 muestras por segundo para que la simulación se lleve a cabo con éxito. Estas 2 muestras son publicadas en DDS y procesadas por el servicio de control.

- Servicio DDS

  El servicio DDS es lanzado por medio de OpenSplice DDS, que puede adquirirse desde la página oficial de Prismtech con una licencia de 30 días de duración. También existe una versión de software libre mucho más limitada, que no se ha utilizado en este trabajo.

  El comando para inicializar el servicio DDS es *osplstart*. Una vez lanzado, es posible monitorizar los topics en tiempo real por medio de la herramienta OpenSplice DDS Tuner. En la Figura 6.14 se puede observar esta aplicación en funcionamiento.

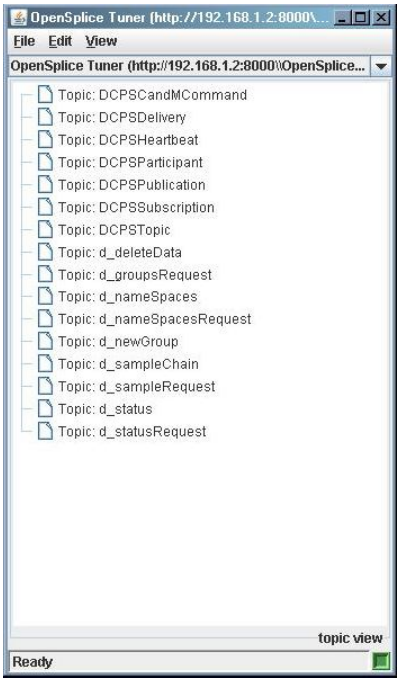

Figura 6.14 OpenSpliceTuner

Inicialmente, al ejecutarse el servicio DDS, OpenSplice presenta una serie de Topics que existen siempre, y son concernientes a la especificación, a la propia implementación, o a servicios externos que pueden añadirse.

La conexión del Tuner con el Global Data Space de DDS puede realizarse de forma local, o remota a través del puerto 8000, manejando SOAP como protocolo estándar. Si se va a realizar de esta última forma, antes de iniciarse el servicio DDS debe crearse el servicio *TunerService* desde la herramienta OpenSplice DDS Configurator, que se muestra en la Figura 6.15.

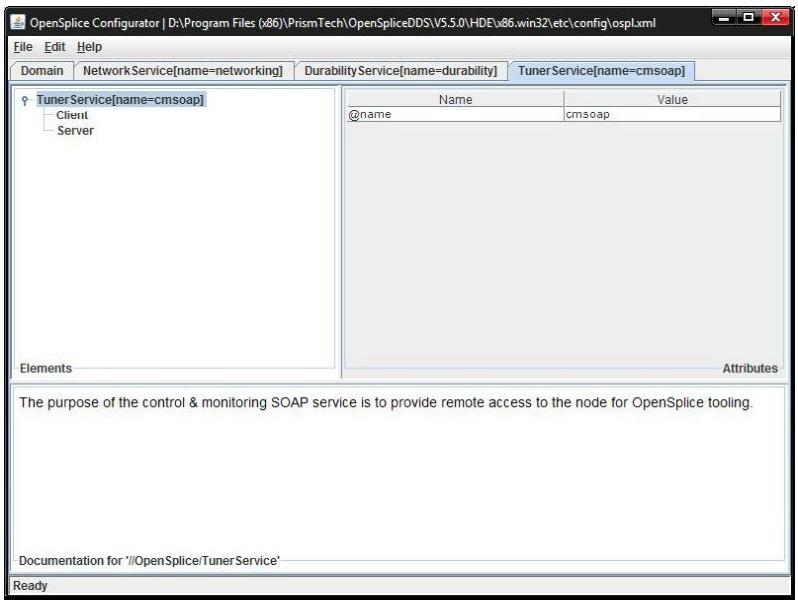

Figura 6.15 OpenSplice Configurator

Estas dos herramientas han sido empleadas de forma extensiva a lo largo de la realización de este trabajo, pero debido a la amplia variedad de funcionalidades presentadas, queda fuera del objetivo de esta memoria presentarlas todas. Puede encontrarse más información de las mismas en la documentación de OpenSplice DDS.

- Servidor web y base de datos

Se ha optado por MySQL como base de datos para el almacenamiento de los datos que van a servir de intermediario entre Matlab y DDS. Asimismo, se considera Apache como servidor Web y el interfaz PHPmyAdmin basado en Web para la gestión de la base de datos.

La instalación de todas estas aplicaciones se realiza por medio de XAMPP [69], que reúne las últimas versiones y presenta un panel control sencillo (Figura 6.16) para iniciar cada uno de ellos.

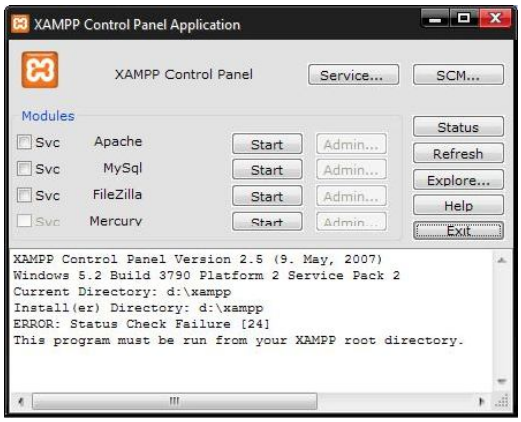

Figura 6.16 Panel de Control de XAMPP

- Entornos de desarrollo

  El entorno de desarrollo empleado para la implementación en Java es PowerTools, que está basado en Eclipse y es suministrado también por OpenSplice como versión de evaluación con una licencia de 30 días.

  A pesar de ser un producto relativamente inestable, debido a su alto consumo de recursos y a sus constantes bloqueos, provee una potencialidad interesante para constituir visualmente dominios DDS, entidades del nivel DCPS y relaciones entre las mismas. Además, permite la generación de código C, C#, C++ y Java, así como la creación de proyectos para Eclipse y Visual Studio.

  En la figura 6.17 puede observarse una captura de este entorno de desarrollo.

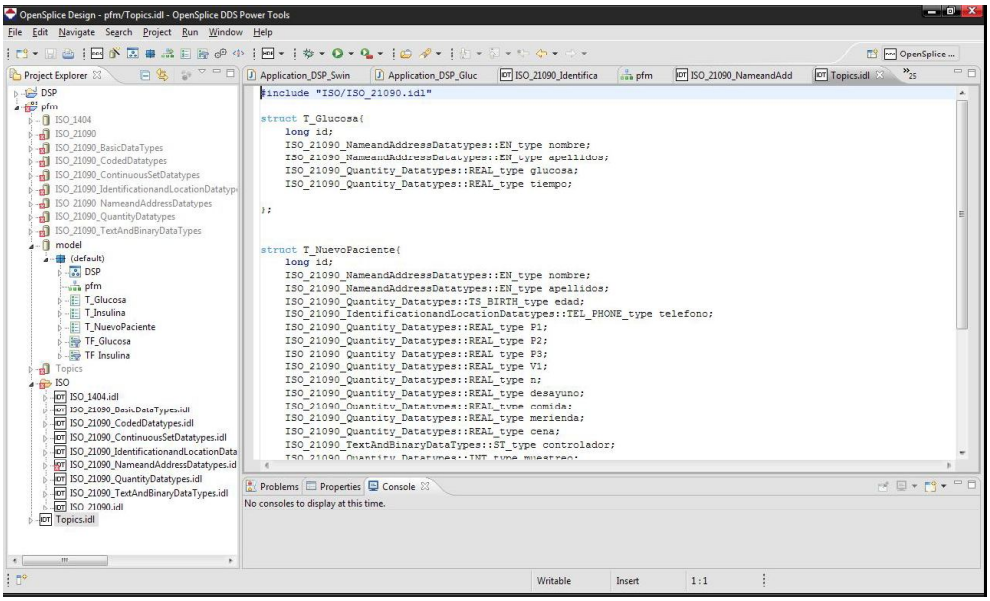

Figura 6.17 PowerTools

Otra de sus ventajas es la capacidad de reunir en un proyecto DDS toda la información pertinente y necesaria de manera muy clara. Asimismo, podemos desarrollar y analizar código IDL, visualizar y/o modificar diferentes modelos de datos compuestos, crear aplicaciones dentro de diagramas del Global Data Space para posteriormente generar código, o configurar políticas de calidad de servicio mediante interfaz gráfico. En general, se trata de una alternativa realmente atractiva para la implementación de aplicaciones DDS.

- Herramientas para la documentación

  Para documentar la implementación y el trabajo realizado, resulta fundamental la creación de diagramas que permitan simplificar de manera visual determinadas tareas que se han llevado a cabo. Es por ello que se ha optado por SmartDraw [70], que es una aplicación poderosa para la creación de diagramas de todo tipo.

  Otro de los elementos importantes es la creación del perfil UML de DDS para el Global Data Space. Este perfil ha sido especificado recientemente [71] por la OMG, y provee un lenguaje de modelado para diseñadores de manera que no se necesite entender la implementación DDS subyacente. Actualmente Enterprise Architect [72] es la única aplicación que provee esta funcionalidad.

6.2.4 Topics en DDS utilizados para el servicio

Para el intercambio de datos entre aplicaciones dentro de DDS, se han construido los topics necesarios a partir de los archivos implementados anteriormente correspondiente a la norma ISO/DES 21090, importando el archivo ISO_21090.idl.

```
#include "ISO/ISO_21090.idl"

struct T_Glucosa{
    long id;
    ISO_21090_NameandAddressDatatypes::EN_type nombre;
    ISO_21090_NameandAddressDatatypes::EN_type apellidos;
    ISO_21090_Quantity_Datatypes::REAL_type glucosa;
    ISO_21090_Quantity_Datatypes::REAL_type tiempo;

};


struct T_NuevoPaciente{
    long id;
    ISO_21090_NameandAddressDatatypes::EN_type nombre;
    ISO_21090_NameandAddressDatatypes::EN_type apellidos;
    ISO_21090_Quantity_Datatypes::TS_BIRTH_type edad;
    ISO_21090_IdentificationandLocationDatatypes::TEL_PHONE_type telefono;
    ISO_21090_Quantity_Datatypes::REAL_type P1;
    ISO_21090_Quantity_Datatypes::REAL_type P2;
    ISO_21090_Quantity_Datatypes::REAL_type P3;
    ISO_21090_Quantity_Datatypes::REAL_type V1;
    ISO_21090_Quantity_Datatypes::REAL_type n;
    ISO_21090_Quantity_Datatypes::REAL_type desayuno;
    ISO_21090_Quantity_Datatypes::REAL_type comida;
    ISO_21090_Quantity_Datatypes::REAL_type merienda;
    ISO_21090_Quantity_Datatypes::REAL_type cena;
    ISO_21090_TextAndBinaryDataTypes::ST_type controlador;
    ISO_21090_Quantity_Datatypes::INT_type muestreo;
    ISO_21090_Quantity_Datatypes::INT_type muestras;
};

struct T_Insulina{
    long id;
    ISO_21090_NameandAddressDatatypes::EN_type nombre;
    ISO_21090_NameandAddressDatatypes::EN_type apellidos;
    ISO_21090_Quantity_Datatypes::REAL_type insulina;
    ISO_21090_Quantity_Datatypes::REAL_type tiempo;
};

#pragma keylist T_NuevoPaciente id
#pragma keylist T_Insulina id
#pragma keylist T_Glucosa id
```

Figura 6.18 Topics construidos para el Servicio DDS

- T_NuevoPaciente: se publica una determinada muestra en este topic cuando un nuevo paciente es añadido al sistema. En este caso debe garantizarse que los datos se entreguen de forma correcta al suscriptor, por lo que la política de calidad de servicio ReliabiltyQosPolicy para este topic debe configurarse como RELIABLE, en lugar de BEST_EFFORT. La diferencia entre ambas radica en que RELIABLE es orientada a conexión, y por tanto, existe un asentimiento entre publicadores y suscriptores por cada muestra que los publicadores generen.
  Los campos que constituyen este topic reúnen todos los datos identificativos del paciente, parámetros del modelo de Bergman, y otro tipo de información que se especifica a través de la interfaz gráfico.

- T_Glucosa: en este topic se publica información generada por el modelo de Bergman que

corre en Matlab, y están suscritos a él el servicio de control y el de alarmas, éste último por medio de un ContentFilteredTopic. De todas formas, la versatilidad de DDS permite que cualquier nuevo elemento de comunicación pueda suscribirse de manera flexible a este topic, por ejemplo para monitorizar la concentración de glucosa de un determinado paciente en particular. Para este caso podría emplearse un ContentFilteredTopic que filtrase el campo *id*.

- T_Insulina: en este topic publica información el servicio de control, ya que éste es el encargado de especificar la cantidad de insulina que debe inyectarse en sangre al paciente para un instante dado a partir de los datos recogidos del topic de glucosa. Obviamente el modelo de Bergman está suscrito a este topic de insulina para completar el camino de retroalimentación, aunque al igual que sucede con el resto de topics, cualquier nueva entidad de comunicación puede suscribirse sin complicación alguna en cualquier momento.

En IDL los topics se construyen a partir de estructuras (*struct*) compuestas por diferentes tipos de datos. Se emplean las partículas *pragma* y *keylist*, seguido del nombre del Topic (que coincide con el nombre de su estructura asociada), y del tipo de dato *key*. El *key* se asimila al concepto de *PRIMARY KEY* en base de datos. En nuestro caso, el *key* va a corresponder con el identificador de usuario (*id*), que se define como un número de tipo entero. En DDS el parámetro *key* no es obligatorio y puede omitirse.

Una particularidad interesante a resaltar es acerca de la cantidad de muestras que deben almacenarse en un determinado topic para suscriptores *late-joining*. Un suscriptor *late-joining* es aquel que se suscribe a un topic cuando ya se ha publicado la información que le interesa. En DDS puede controlarse este aspecto gracias el desacoplamiento intrínseco entre entidades. La política de calidad de servicio DurabilityServiceQoSPolicy permite configurar el número de muestras que pueden almacenarse en un determinado topic para suscriptores late-joining, permitiendo que puedan almacenarse todas las que se publican (KEEP_ALL_HISTORY_QOS) o sólo un número específico de ellas (KEEP_LAST_HISTORY_QOS y los parámetros history_depth, maxsamples, max_instances). Debe tenerse en cuenta que esta configuración es establecida a la hora de crear el topic, ya que una vez que está constituido no es posible cambiar los parámetros. En DDS no existe posibilidad de borrar el topic y crearlo de nuevo, ya que el tiempo de vida de un topic una vez creado no finaliza hasta que lo hace el servicio DDS. La única solución viable es crear un nuevo topic con nombre distinto y una nueva política de calidad de servicio.

Por último, comentar que existe un topic especial denominado ContentFilteredTopic que está asociado a un determinado topic convencional y que permite la sustracción de datos a partir de su contenido. Este tipo de topics especiales no se implementan en los archivos IDL, sino que son creados directamente desde el código a partir de la función *create_contentfilteredtopic*, pasándole como parámetro el topic al que se le asocia y una sentecia SQL que restringe los datos en función de su contenido. Por ejemplo, el servicio de alarmas está suscrito a un ContentFilteredTopic vinculado al topic de glucosa (T_Glucosa) cuya restricción SQL es “glucosa <70 or glucosa >140”, por lo que dicho servicio únicamente recibiría muestras cuando los valores de concentración de glucosa para un determinado paciente sean anormales. Si el servicio de alarmas estuviera suscrito directamente al topic de glucosa (T_Glucosa), tendría que discriminar la información directamente él mismo, suponiendo una carga computacional innecesaria. Los ContentFilteredTopic son una funcionalidad realmente potente en DDS, transparente al desarrollador, y que evita transmisiones redundantes de información.

6.2.5 Base de datos

Para este trabajo, las tablas empleadas en la base de datos son muy sencillas y no guardan ningún tipo de relación entre las mismas, por lo que únicamente se expondrán cada una de ellas y los diferentes campos en los que se compone en la Figura 6.19.

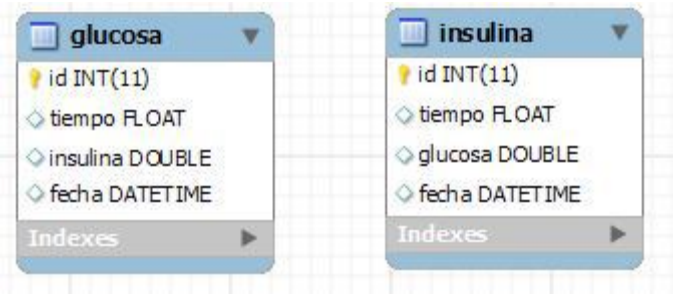

Figura 6.19 Tablas en la base de datos

Estas tablas sirven de intermediario entre el modelo de Bergman y el Global Data Space de DDS. De esta forma, Simulink inserta los valores de la concentración de glucosa del paciente para cada instante, y una aplicación del paciente, implementada en Java, recoge estos valores, los publica en un topic DDS, y elimina las muestras de la base de datos.

El servicio de control implementado en Java recoge las muestras de glucosa del topic en DDS y las inserta en la tabla glucosa de la base de datos local. A su vez, esta información que se va almacenando en la base de datos, es recogida por Simulink, que está ejecutando un determinado algoritmo de control, obteniendo como salida una determinada concentración de insulina que es la que se inyecta al paciente. Simulink inserta estos datos en la tabla insulina, y el servicio de control se encarga de recoger la información y publicarla en el topic de insulina en DDS.

Por último, la aplicación del paciente recoge estos valores de insulina desde DDS y los inserta en la tabla insulina para que el modelo de Bergman pueda utilizarlos. A partir de ahí, el proceso vuelve a empezar.

En la Figura 6.20 se puede contemplar con mayor claridad lo comentado con anterioridad.

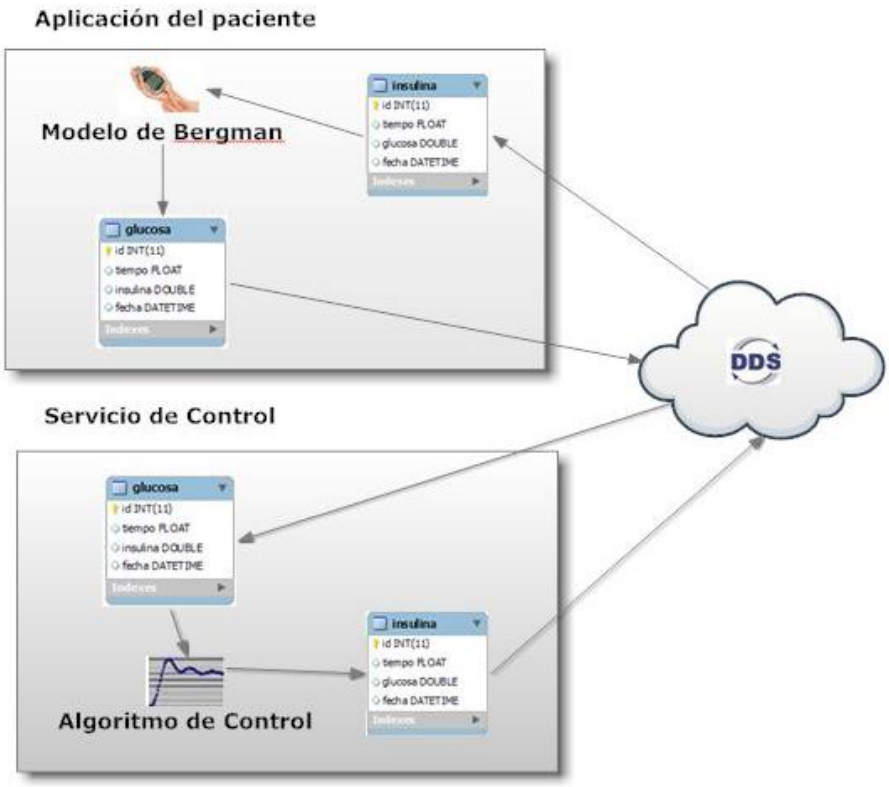

Figura 6.20 Relación entre DDS y la base de datos

6.2.6 Perfil UML para el Global Data Space en DDS

El diseño de un sistema basado en DDS bajo el perfil UML [87] puede ser descompuesto en los siguientes pasos:

1. Definir el modelo de información: describe los tipos de datos que va a contener los topics, utilizando tipos *structs*, *arrays*, *unions* etc., y asocia esta información a un topic con unas políticas de calidad de servicio que van a heredar los DataWriters y DataReaders que publiquen o se suscriban a dicho topic. Este modelo de información ya ha sido detallado en el apartado de metodología del modelo de información y se presenta en la Figura 6.21.

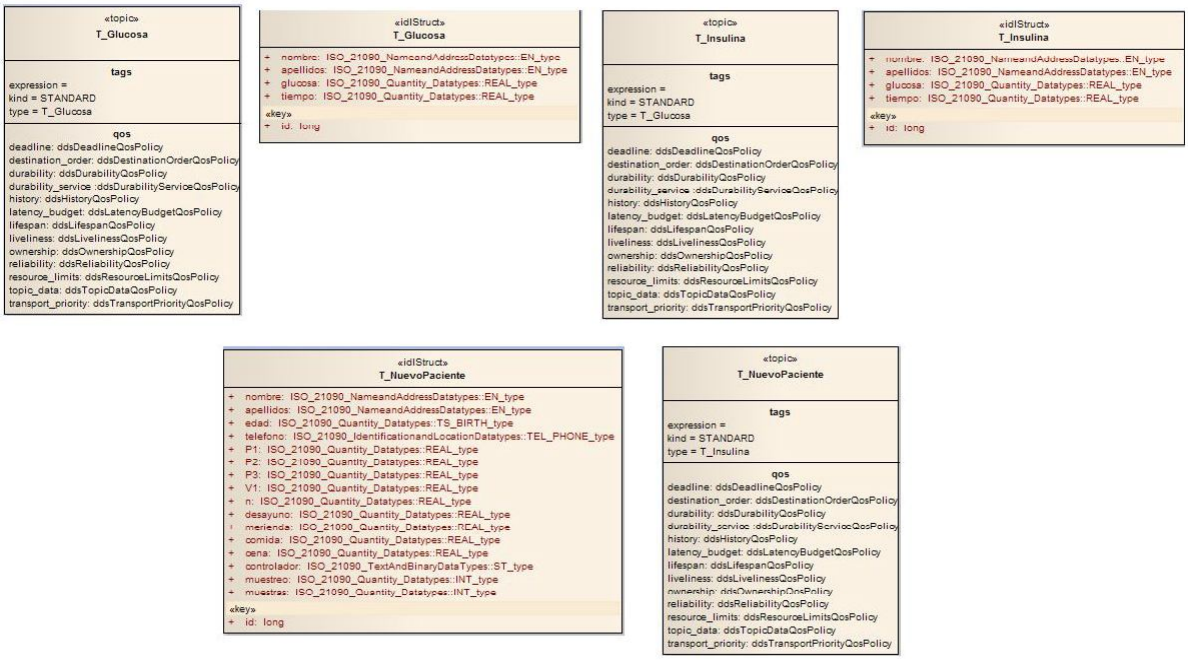

Figura 6.21 Modelo de datos

2. Asociar la QoS con el modelo de información: cada topic lleva asignado una serie de políticas de calidad de servicio que son configuradas desde la pestaña “TaggedValues” en QoSProperty. Para el presente trabajo se han utilizado las políticas las QoS por defecto que adoptan los topics en su creación. La parametrización de la QoS se realiza de forma gráfica como puede contemplarse en la Figura 6.22.

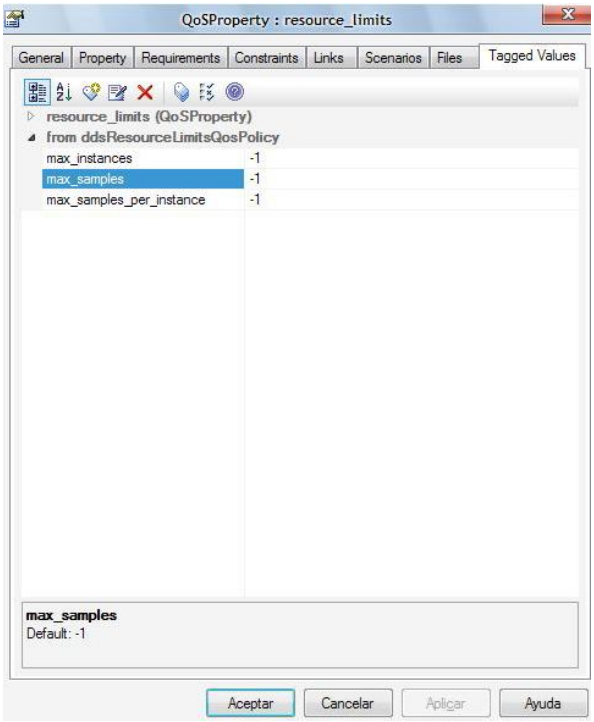

Figura 6.22 QoS para ResourceLimitQoSPolicy

3. Definir Particiones y Dominios: el nombre del dominio es el de “OpenSpliceV5.5.0”, que es

el que toma por defecto el Global Data Space cuando se arranca el servicio DDS. Las particiones permiten realizar subdivisiones lógicas dentro de un dominio para aislar diferentes funcionalidades. Para nuestro caso no se ha definido ninguna partición.

4. Identificar DataWriters y DataReaders: los vínculos entre Topics, DataWriters y DataReaders ya ha sido comentada en apartados anteriores, por lo que únicamente se expone el resultado final en lenguaje UML en la figura 6.23.

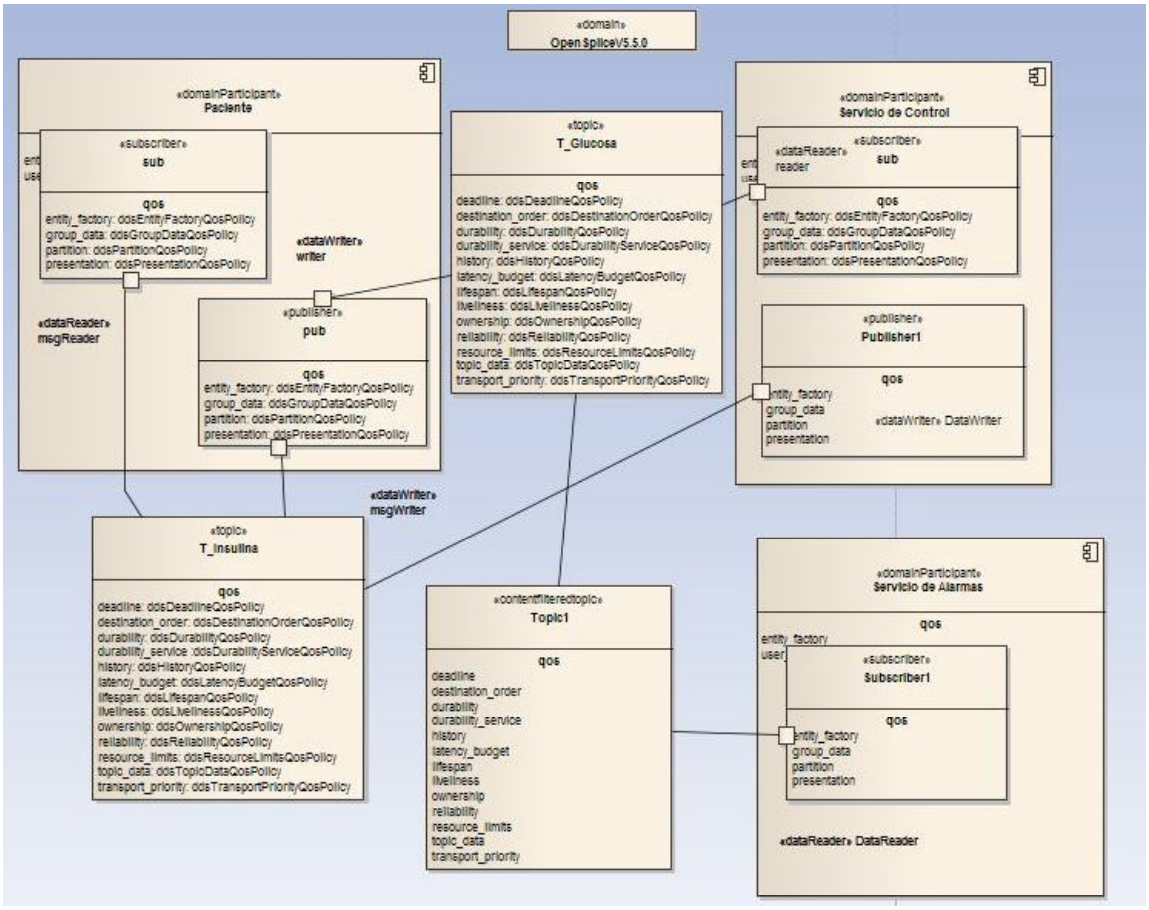

Figura 6.23 Perfil UML del Global Data Space

5. Definir requisitos QoS para los DataWriters/DataReaders: al igual que en el paso número 2, las políticas de calidad de servicio para DataWriters/DataReaders han sido configuradas por defecto en su creación.

6. Unir el modelo creado a un específico PSM: un PIM provee una plataforma independiente de entidades de modelos que es mapeada a una implementación específica o plataforma determinada (PSM). Para este caso, el PSM definiría el lenguaje de programación Java bajo el entorno de programación Eclipse para el desarrollo de aplicaciones en entornos con una arquitectura de 64 bits, software OpenSplice DDS, y sistema operativo Windows. Por último, se generaría el código correspondiente.

6.2.7 Manual de usuario

Antes de ejecutarse la aplicación, debe iniciarse la base de datos MySQL desde el panel de control de XAMPP. Una vez realizado esto, puede ejecutarse la aplicación, presentando posteriormente una serie de servicios que se ilustran en la Figura 6.24:

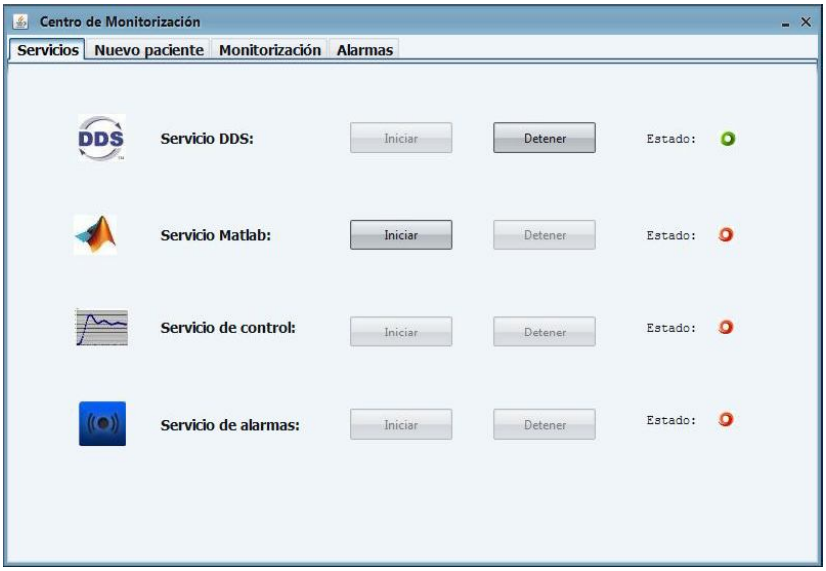

Figura 6.24 Servicios

- Servicio DDS: debe de iniciarse si quiere llevarse a cabo la comunicación entre procesos y aplicaciones mediante DDS.
- Servicio Matlab: este servicio lanza una primera instancia Matlab para preparar a la aplicación de posibles llamadas a las funciones de Matlab. Debido a que el lenguaje de Matlab no es concurrente, la aplicación lanza automáticamente una instancia Matlab por cada nuevo paciente que se crea. Cuando este servicio y el de DDS están activos, es posible acceder a las pestañas de nuevo paciente y monitorización.
- Servicio de control: si no se activa este servicio, la concentración de insulina inyectada al paciente es 0, ya que no existe acción de control. No es posible activar este servicio si no están activos antes el servicio DDS y el servicio Matlab.
- Servicio de alarmas: notifica cuando un determinado paciente alcanza valores de glucosa dentro de los estados de hiperglucemia o hipoglucemia. No es posible activar este servicio si no están activos antes el Servicio DDS y el Servicio Matlab. Cuando este servicio está activo, es posible acceder a la pestaña de alarmas

En la pestaña nuevo paciente vamos a encontrarnos con un formulario a rellenar para cada paciente nuevo que queramos añadir, tal y como se muestra en la Figura 6.25.

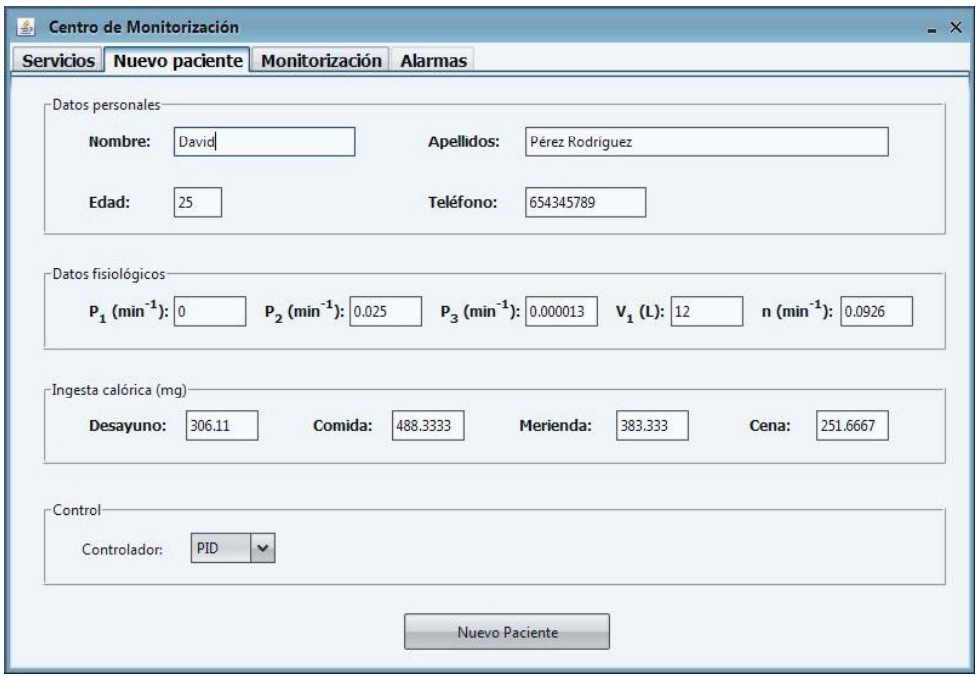

Figura 6.25 Nuevo Paciente

Los datos a introducir para un determinado paciente son:

- Datos personales: que no influyen a la hora de ejecutar el modelo.
- Datos fisiológicos: relacionados con los parámetros utilizados en el modelo de Bergman. Los valores por defecto son los correspondientes a los de una persona diabética de tipo 1 con peso normal.
- Ingesta calórica: cantidad de calorías en mg que el paciente ingiere en cada comida.
- Control: pueden configurarse diferentes tipos de controladores: PID, H-infinito, LQR, LTR y QFT.

Una vez añadido el nuevo paciente, se abrirá automáticamente una nueva ventana en donde se mostrará los valores de la concentración de glucosa para cada instante durante los 1200 minutos que dura la simulación, como se aprecia en la Figura 6.26.

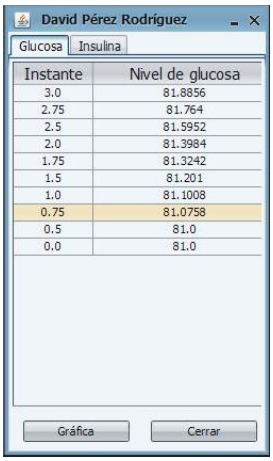

Figura 6.26 Valores de glucosa para un paciente

Si pulsamos sobre el botón Gráfica, se visualizará la evolución en tiempo real de la concentración de glucosa en sangre para cada instante del paciente en cuestión, tal y como se observa en la Figura 6.27.

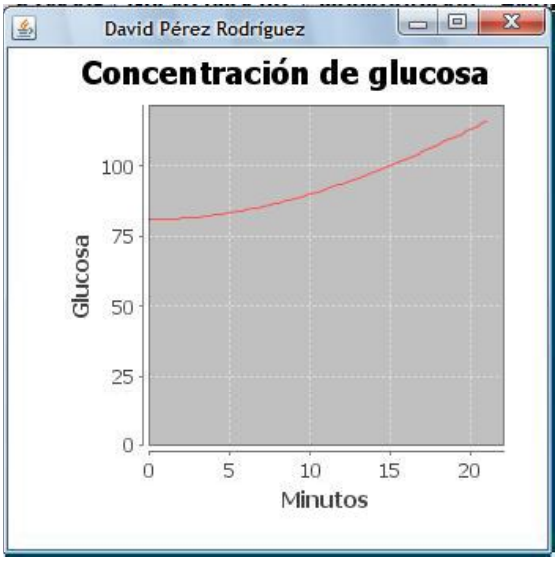

Figura 6.27 Evolución de la concentración de glucosa en tiempo real

Si el Servicio de Control está inactivo, la acción de control es nula, y por tanto la concentración de insulina inyectada es 0. Si el Servicio de Control está activo, los valores de glucosa se encontrarán dentro del rango permitido. Los resultados variarán en función de los parámetros del modelo, la cantidad de comida ingerida por parte del paciente, y del controlador seleccionado.

En la pestaña de monitorización, la concentración de glucosa se va actualizando con el tiempo y se muestra su valor para el momento actual, como se presenta en la Figura 6.28.

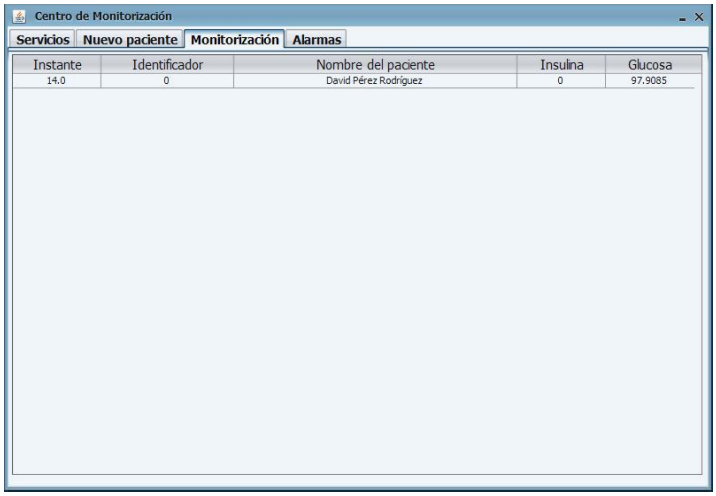

Figura 6.28 Monitorización de glucosa

El Servicio de Alarmas también alerta de posibles valores anormales de glucosa para los pacientes. Concretamente cuando se alcanzan valores por encima de 140 mg/dL (estado de hiperglucemia) y por debajo de 70 mg/dL (estado de hipoglucemia). Esto se contempla en la Figura 6.29.

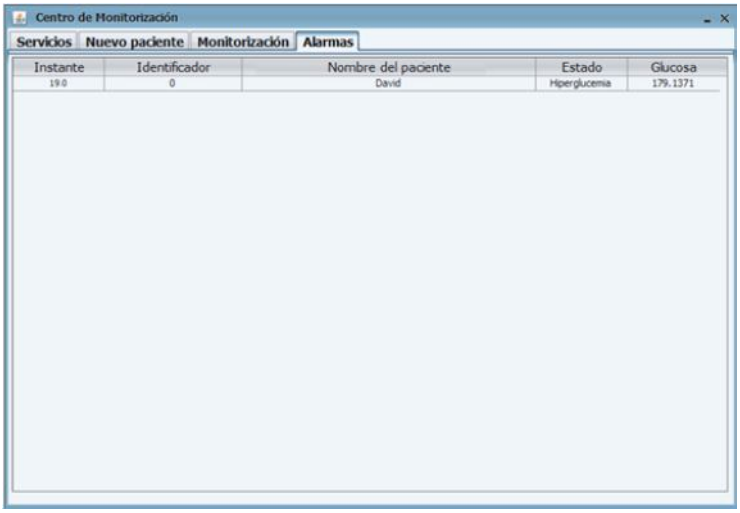

Figura 6.29 Sistema de Alarmas

6.2.8 Manual del programador

Implementación en Matlab/Simulink

No es objetivo de esta memoria detallar cada uno de los distintos controladores implementados. Para una descripción de los mismos, puede acudirse a la memoria del trabajo de la asignatura de Control Robusto.

El modelo de Bergman es contemplado en este trabajo como una “caja cerrada”, únicamente considerándose los valores de entrada y los de salida. Estos valores son adquiridos y suministrados a la base de datos para una posterior publicación/suscripción a DDS. Para llevar a cabo esta labor es necesaria la adición de bloques *MatlabFunction*, que permitan la inserción de código en la simulación de Simulink. En la Figura 6.30 se aprecian dos bloques de este tipo, uno de recogida de datos a la entrada, y otro de inserción a la salida.

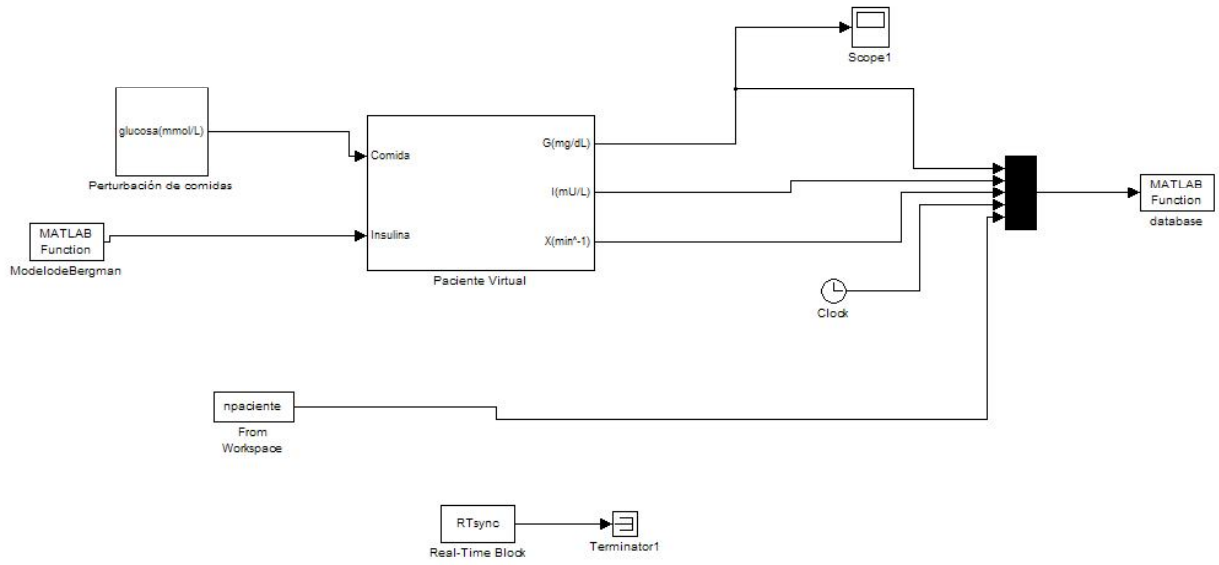

Figura 6.30 Modelo de Bergman

En la Figura 6.31 se muestra el interior del Paciente Virtual, donde se multiplexan todos los parámetros recogidos de la Aplicación de Java y se suministra toda esta información a otra función de Matlab, donde se encuentra implementado el modelo.

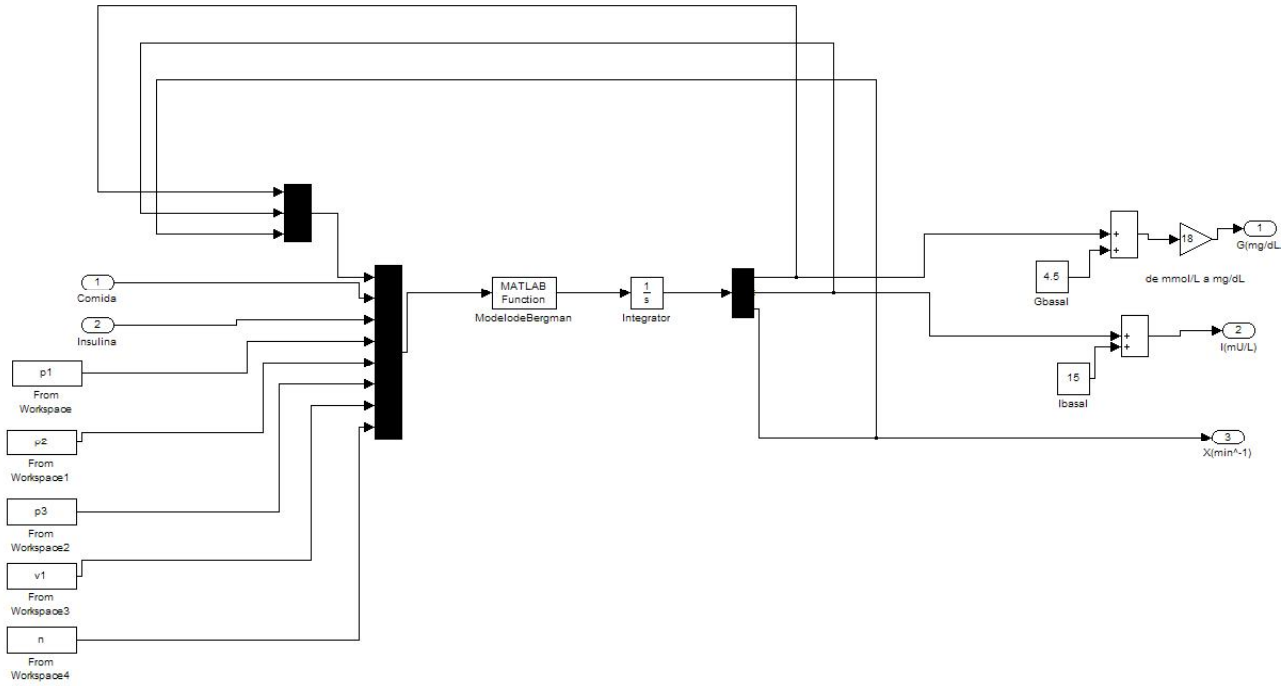

Figura 6.31 Paciente Virtual

Implementación en Java

Como se vio anteriormente, en la configuración del Global Data Space cada DomainParticipant corresponde con una aplicación que está ejecutándose de forma independiente y que utiliza DDS como mecanismo de intercambio de información.

De esta forma, tendríamos 3 aplicaciones. Por un lado el DSP o aplicación del cliente, cuyo número de instancias es indeterminado y depende del número de pacientes existentes en el sistema (una aplicación por cada paciente), y dos aplicaciones que conciernen al Servicio de Control y el Servicio de Alarmas, ambos únicos.

Con motivo de simplificar la aplicación y agilizar el proceso de pruebas, el Global Data Space resultante definido en PowerTools va a reunir todos los DomainParticipant en una sola aplicación. De esta forma, cada DomainParticipant correspondería a un proceso o hilo en Java que es creado siguiendo el paradigma de programación concurrente en Java. La comunicación entre procesos sería utilizando DDS.

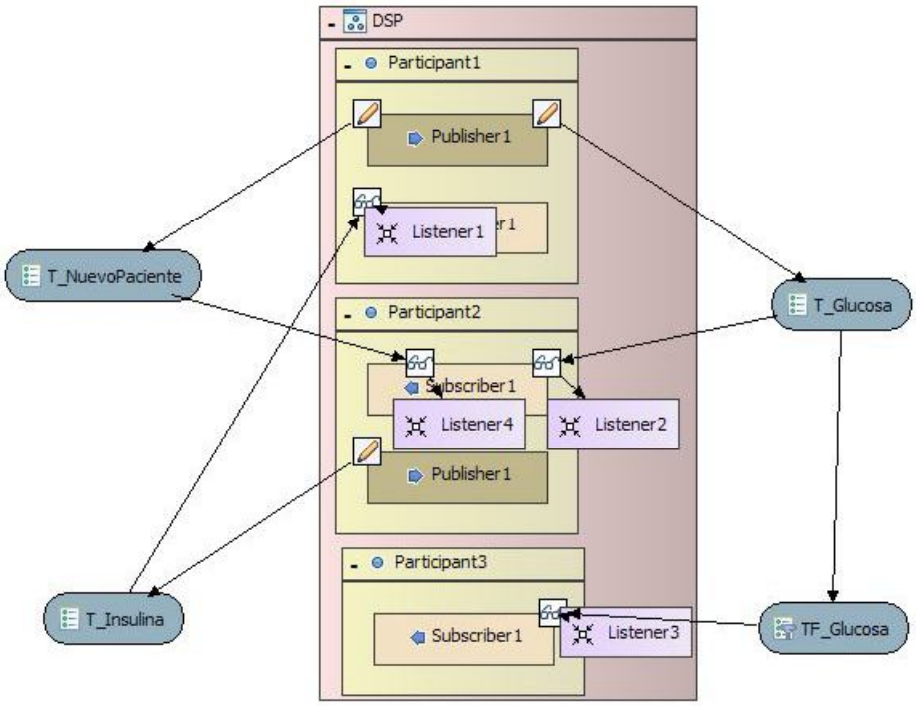

Figura 6.32 Global Data Space en PowerTools

Como puede apreciarse en la figura 6.32, la configuración final del Global Data Space es muy similar a la ya expuesta en apartados anteriores. La única diferencia notable radica en la incorporación de los elementos Listener, que se adhieren a cada DataReader y actúan como disparador o trigger para los mismos cuando una nueva muestra es publicada dentro de un Topic a los que los Datareaders están suscritos. De esta manera, cuando una nueva muestra es publicada, el Listener llama automáticamente dentro del DataReader a la función *on_data_available* para un posterior tratamiento de la información recibida.

Una entidad similar al Listener es el WaitSet. La diferencia entre ambas reside en que un Waitset es un mecanismo bloqueante, es decir, la aplicación se detiene hasta que una nueva muestra es recibida. En este caso se ha optado por el Listener porque no se pretende que la aplicación se detenga bajo ningún concepto.

Una vez diseñado el diagrama anterior, se genera de forma sencilla el código resultante en Java. Lamentablemente PowerTools es una herramienta relativamente reciente e inestable, por lo que presenta deficiencias a la hora de llevar a cabo este cometido. Es necesario por tanto realizar

una serie de modificaciones posteriores sobre el código.

Los fallos de generación de código se encuentran en la creación del ContentFilteredTopic, y más concretamente en la función *create_contentfilteredtopic*, donde no se especifica explícitamente los parámetros de la restricción SQL, ni tanto el Topic al que debe asociarse. También existen problemas con la creación de los Listeners y la adhesión de los mismos a las entidades DataReaders.

A continuación se describe brevemente cada una de las clases Java implementadas, que son independientes del código anterior generado:

- **Application_Alarmas:** es uno de los threads o hilos únicos, y una vez creado su única funcionalidad es suscribirse al ContentFilteredTopic de glucosa mediante una llamada a *DDS_AlarmasSubscribeGlucosa*. Cuando llega una muestra, se salta al método *on_data_available*, que se encuentran en la clase *DDS_AlarmasSubscribeGlucosaListener*.

- **Application_Control:** es otro thread o hilo único, y únicamente está suscrito al topic de glucosa mediante la llamada a *DDS_ControlSubscribeGlucosa*. Al igual que en el caso anterior, cuando llega una nueva muestra, se realiza automáticamente una llamada a 1*on_data_available* por medio del Listener asociado, y que se encuentra en la clase *DDS_ControlSubscribeGlucosaListener*.

- **Application_DSP_GlucosaGrafica:** va dibujando la gráfica de los valores de concentración de glucosa a lo largo del tiempo que va suministrando el modelo en Simulink. Para ello se utiliza la librería externa de Java Jfree [73].

- **Application_DSP_GlucosaGraficaListener:** en esta clase se implementa los métodos disparadores que se encuentran en la ventana de la gráfica de glucosa.

- **Application_DSP_InsulinaGrafica:** va dibujando la gráfica con los valores de concentración de insulina a lo largo del tiempo que va adquiriendo el modelo en Simulink.

- **Application_DSP_InsulinaGraficaListener:** en esta clase se implementa los métodos disparadores que se encuentran en la ventana de la gráfica de insulina.

- **Application_DSP_Swing:** es una clase inicializada por *Application_DSP_Swing*, y crea las dos pestañas JTabbedPane para los *JPane* que crean *Application_DSP_SwingGlucosa* y *Application_DSP_SwingInsulina*.

- **Application_DSP_SwingGlucosa:** es una clase inicializada por *Application_DSP_Swing* y crea el *JPane* en donde se sitúan la JTable con los valores de glucosa y los botones de Cerrar y Gráfica.

- **Application_DSP_SwingGlucosaListener:** implementa los disparadores para los botones de Cerrar y Gráfica que son creados en *Application_DSP_SwingGlucosa*.

- **Application_DSP_SwingInsulina:** es una clase inicializada por *Application_DSP_Swing* y crea el *JPane* en donde se sitúan la JTable con los valores de insulina y los botones de Cerrar y Gráfica.

- **Application_DSP_SwingInsulinaListener:** implementa los disparadores para los botones

de Cerrar y Gráfica que son creados en *Application_DSP_SwingInsulina*.

- **Application_DSP:** es otro de los thread o hilo único y uno de los más importantes. Se encarga de llamar a la función de Matlab *run_bergmander* por medio de la creación de un objeto de la clase *Application_Matlab_ReturningEval* y pasándole como parámetro todos los valores correspondientes al modelo, así como las cantidades de comida ingerida por el paciente y el tipo de controlador que va a emplearse.
  También va a encargase de llamar a la interfaz gráfico por medio de la clase DSP_Swing, en donde se va visualizando en tiempo real los valores de glucosa para cada instante.
  Por último, esta clase adquiere de la base de datos los valores de glucosa que van generando como salida la simulación en Simulink, y los va publicando a medida que llegan al Topic correspondiente de glucosa mediante el método estático *send* de la clase *DDS_DSPPublishGlucosa*.

- **Application_Matlab_ReturningEval:** es el interfaz de comunicación con Matlab y hace uso del API *matlabcontrol* por medio de un array de *proxys* que se van creando con cada nueva simulación que se necesita correr en Matlab. El objeto *proxy* pertenece a la clase *RemoteMatlabProxy* que forma parte de *matlabcontrol*, y para la invocación remota de comandos Matlab se hace uso del método *returningEval*, que permite retornar el resultado que devuelve Matlab.

- **BD:** es la clase encargada de la comunicación con la base de datos. Posee un atributo privado estático que se inicializa cuando se ejecuta la aplicación principal, y es empleado en la invocación de cada uno de los métodos estáticos que componen la clase. La conexión con la base de datos se realiza mediante el driver JDBC (Conectividad de Base de Datos Con Java), que es el estándar de comunicaciones en Java para conectar con base de datos. Concretamente se hace uso del driver JDBC para MySQL. La conexión se realiza en el equipo local a través del puerto por defecto 3306.

- **ColaPacientes:** es una clase muy simple que únicamente posee un método que es invocado cuando se pretende obtener un nuevo identificador para un paciente que va a añadirse al sistema.

- **Content_Alarmas:** crea el *JPane* correspondiente a la pestaña Alarmas, y por tanto está compuesto de una *JTable* en donde se notifica de los pacientes que poseen valores anormales de glucosa.

- **Content_Monitorizacion:** crea el *JPane* correspondiente a la pestaña Monitorización, y por tanto está compuesto de una *JTable* con todos los pacientes existentes en el sistema, así como los valores de glucosa para cada uno de ellos en el instante actual.

- **Content_NuevoPaciente:** crea el *JPane* correspondiente a la pestaña NuevoPaciente e incluye todos los elementos Swing del formulario que es necesario completar para añadir un nuevo usuario al sistema.

- **Content_NuevoPacienteSend:** realiza una validación de todos los campos que se introducen en el formulario de NuevoPaciente y se publica en DDS por medio de la invocación del método *send* de la clase *DDS_DSPPublishNuevoPaciente*.

- **Content_Servicios:** crea el *JPane* desde donde se activan o detienen los diferentes servicios de los que consta la aplicación.

- **Content_ServiciosListener:** implementa los disparadores de los botones que se encuentran en el *JPane* creado por *Content_Servicios.*

- **DDS_AlarmasSubscribeGlucosa:** únicamente adjunta el Listener de la clase DDS_AlarmasSubscribeGlucosaListener correspondiente al DataReader que está suscrito al ContentFilteredTopic de glucosa.

- **DDS_AlarmasSubscribeGlucosaListener:** implementa el método *on_data_available* para el tratamiento de las nuevas muestras que se van recibiendo.

- **DDS_ControlPublishInsulina:** implementa el método *send,* que recibe un objeto de tipo *Topic_Insulina,* y publica en el Topic correspondiente a través del método *write,* que pertenece al DataWriter.

- **DDS_ControlSubscribeGlucosa:** únicamente adjunta el Listener de la clase *DDS_ControlSubscribeGlucosaListener* correspondiente al DataReader que está suscrito al Topic de glucosa.

- **DDS_ControlSubscribeGlucosaListener:** implementa el método *on_data_available* para el tratamiento de las nuevas muestras que se van recibiendo.

- **DDS_DSPPublishGlucosa:** implementa el método *send,* que recibe un objeto de tipo *Topic_Glucosa* y lo publica en el Topic correspondiente a través del método *write*, que pertenece al DataWriter.

- **DDS_DSPPublishNuevoPaciente:** implementa el método *send,* que recibe un objeto de tipo *Topic_NuevoPaciente* y publica en el Topic correspondiente a través del método *write,* que pertenece al DataWriter.

- **DDS_DSPSubscribeInsulina:** únicamente adjunta el Listener de la clase *DDS_DSPSubscribeInsulinaListener* correspondiente al DataReader que está suscrito al Topic de insulina.

- **DDS_DSPSubscribeInsulinaListener:** implementa el método *on_data_available* para el tratamiento de las nuevas muestras que se van recibiendo.

- **DSP:** es la clase principal de la aplicación, y únicamente se encarga de crear un nuevo objeto de la clase *Swing_MainWindow.*

- **ModelTable:** es el modelo de la tabla que utiliza el elemento *JTable* de Swing, y hereda los atributos y métodos de la clase de Java *DefaultTableModel*.

- **Services_Alarmas:** inicia el servicio de Alarmas mediante la activación de la pestaña Alarmas.

- **Services_Control:** crea un nuevo proxy de conexión con Matlab para ejecutar los distintos controladores que van a emplearse.

- **Services_DDS:** es la clase que inicializa el servicio DDS, y contiene la ruta en donde se encuentra instalado OpenSplice.

- **Services_Matlab:** inicializa el servicio de Matlab mediante la creación de un objeto de tipo RemoteMatlabProxyFactory, y la invocación del método *getProxy* para la adquisición de un proxy de conexión con Matlab. A continuación, se declaran todas las variables que vayan a utilizarse en Matlab y Simulink como globales por medio del método *eval*. De esta forma se consigue que el modelo en Simulink pueda adquirir las variables pertinentes que se hallan en el workspace de Matlab.

- **Swing_MainWindow:** implementa la ventana principal de la aplicación, y a continuación crea un objeto de la clase *Swing_Swing_Tabs*.

- **Swing_MainWindowListener:** implementa los disparadores para la ventana principal, como por ejemplo, las acciones a llevar a cabo cuando se cierra la aplicación.

- **Swing_SwingTabs:** implementa las pestañas de cada uno de los servicios mediante el uso del elemento *JTabbePane* y va llamando a las clases *Content_Servicios*, *Content_NuevoPaciente*, *Content_Monitorizacion*, *Content_Alarmas* que implementan el contenido de cada una de las pestañas.

- **Topic_Glucosa:** es una clase de tipo JavaBeans cuyo único objetivo es el encapsulamiento de los tipos de datos que conforman el Topic de glucosa en lenguaje Java.

- **Topic_Insulina:** es una clase de tipo JavaBeans cuyo único objetivo es el encapsulamiento de los tipos de datos que conforman el Topic de insulina en lenguaje Java.

- **Topic_NuevoPaciente:** es una clase de tipo JavaBeans cuyo único objetivo es el encapsulamiento de los tipos de datos que conforman el Topic de NuevoPaciente en lenguaje Java.