UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

UNIVERSIDAD POLITÉCNICA DE VALENCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Editor gráfico de correspondencias de alto nivel para arquetipos de Historia Clínica Electrónica

PROYECTO FIN DE CARRERA

Autor: Luis Marco Ruiz
Directora: Montserrat Robles Viejo
Codirector: J. Alberto Maldonado Segura

Valencia, septiembre 2010

AGRADECIMIENTOS

A Jose Alberto Maldonado y Montse Robles por su inestimable ayuda y consejos durante la realización de este proyecto.

A Diego Boscá que siempre me ha ayudado con los aspectos más técnicos del trabajo.

Al resto de mis compañeros del grupo de Ingeniería de Sistemas de Información de IBIME: Carlos Angulo, David Moner y Ernesto Reig.

A mis padres por su apoyo y consejo en todas mis decisiones.

Tabla de contenido

Índice de Figuras:

1. Introducción

1.1 Prólogo

Los centros médicos e instituciones sanitarias manejan gran cantidad de información, por tanto la gestión eficaz y eficiente de la información es de una imperiosa necesidad. Los ingentes volúmenes de datos que requiere procesar la atención médica ha hecho que las Tecnologías de la Información (TI) sean la única solución.

Si bien se ha avanzado enormemente en el eficaz tratamiento de la información sanitaria, por ejemplo, los numerosos esfuerzos dedicados a la puesta en marcha de historias clínicas electrónicas (HCE), un vistazo más cercano nos muestra que estas no están más que en su adolescencia. Prueba de ello son, por ejemplo, los actuales EHR (**E**lectronic **H**ealth **R**ecord) que han llegado a agilizar y abaratar considerablemente los costes en la gestión exclusivamente en el ámbito local (departamento hospitalario y en el mejor de los casos a nivel hospitalario). El problema se sitúa ahora en un ámbito más global donde la ausencia de estándares o la falta de consenso en la aplicación de estos ha provocado que los sistemas locales no sean más que pequeñas islas incomunicadas a las que les resulta imposible el intercambio de información clínica de una forma automatizada con sus vecinas.

Ejemplo de lo anterior son los problemas surgidos con la llamada Interoperabilidad Semántica (IopS). Un paciente puede recibir tratamiento en un hospital y que le sean realizados unos análisis para el diagnóstico de una determinada patología, unos días más tarde, ya habiendo recibido el alta, podría sufrir una crisis y ser derivado por el servicio de urgencias a un segundo centro médico de la misma área. Este segundo centro no tendría acceso a las pruebas ni al diagnóstico que el primero dio. Se vería así obligado a repetir el proceso perdiendo un tiempo que en muchos casos puede resultar vital para la recuperación del paciente y duplicando los gastos asociados a analíticas y diagnóstico. Todo esto sería evitable si los sistemas de los centros fueran semánticamente interoperables. Esto es, si el sistema de información del segundo centro pudiera acceder al del primero recuperando los datos del paciente junto a todas las pruebas y observaciones que le fueron realizadas.

Vemos con el ejemplo anterior cuánto queda por hacer para conseguir un intercambio transparente de la información sanitaria entre los distintos EHRs de cada centro médico. La presente memoria muestra una pequeña aportación a una de las herramientas que intentan hacer que esas islas que hoy en día son los distintos sistemas de información hospitalarios puedan comunicarse de una manera eficaz, transparente y con un tiempo mínimo mejorando así el tiempo y la calidad del diagnóstico. En concreto, el desarrollo de una *interface* visual para facilitar la especificación de correspondencias de alto nivel declarativas (*mappings*) entre los

esquemas de las fuentes de datos ya existentes y definiciones formales de conceptos clínicos basadas en normas para la comunicación de EHR. De tal forma, que la información contenida en estas fuentes de datos pueda ser transformada (reestructurada y etiquetada) para ser compatible con alguna norma (o formato propietario) que facilite su comunicación con otro sistema de información.

1.2. Grupo de Investigación

El Proyecto Final de Carrera ha sido llevado a cabo en el grupo IBIME (Informática Biomédica). IBIME es un grupo de investigación englobado en el ***I**nstituto de **A**plicaciones de las **T**ecnologías de la **I**nformación y las **C**omunicaciones **A**vanzadas* (ITACA) en la Universidad Politécnica de Valencia. El grupo tiene una estructura multidisciplinar que abarca las áreas:

Imagen Médica: estudio de segmentación y representación de imágenes en 3D, mapas cerebrales de perfusión, difusión y funcionalidades en Resonancia Magnética Nuclear.

Minería de Datos Biomédicos: cubre las técnicas de reconocimiento de patrones, modelado, predicción computacional y desarrollo de herramientas para el procesado de datos biomédicos. Esta línea de investigación ha participado y participa en proyectos europeos de gran relevancia en el campo de sistemas médicos de ayuda a la toma de decisión como por ejemplo ***e-tumor***. A su vez su software **Curiam** se encuentra en fase de implantación en el Hospital Peset y la clínica Quiron, ambos de Valencia.

Ingeniería de la Información Biomédica: abarca la integración y normalización de datos biomédicos procedentes de sistemas distribuidos y bases de datos heterogéneas en una vista de datos integrada para obtener la Historia Clínica Electrónica federada aplicando el estándar CEN/TC 251 EN13606.

Es en esta última área donde se ha desarrollado el proyecto aquí descrito. Esta línea participa actualmente en diversos proyectos de investigación sobre interoperabilidad semántica en salud. El trabajo descrito en la presente memoria se encuadra dentro del proyecto "Plataforma para la adquisición y compartición de información y conocimiento para comunidades de investigación clínica en red" financiado por el Ministerio de Educación y Ciencia dentro del Plan Nacional de I+D+i, referencia TSI2007-66575-C02-01. Otros proyectos desarrollados por el grupo de investigación relevantes al ámbito de este proyecto fin de carrera. Cabe destacar, son el sistema de integración de datos ***Pangea***, que ha permitido la implantación de un sistema federado visor de Historia Clínica Electrónica en el Consorcio Hospital General Universitario de Valencia, y de desplegar un sistema que permite integrar el Hospital Universitario de Fuenlabrada con sus centros de atención primaria. Con este último proyecto se ha logrado la conciliación de la medicación de los pacientes entre centros de atención primaria y especializada.

1.3. Comités y organismos relacionados

Son numerosos los organismos nacionales e internacionales involucrados en el avance y desarrollo de la interoperabilidad semántica en el sector sanitario.

A nivel nacional, el principal coordinador es el Ministerio de Sanidad. Es este ministerio el que financia proyectos de investigación que permitan el desarrollo de metodologías y tecnologías relacionadas con la interoperabilidad de sistemas sanitarios. A su vez, se encarga de coordinar esfuerzos y marcar las metas a conseguir en los años sucesivos.

A nivel europeo, es el *Semantic Interoperability Centre Europe* (SEMIC) el encargado de coordinar las iniciativas de los distintos países en materia de interoperabilidad semántica. Su objetivo es hacer uso del conocimiento ya generado en proyectos previos para el beneficio de otros posteriores. A su vez EUROREC, una organización europea sin ánimo de lucro, es el encargado de promocionar el uso de EHR altamente fiables definiendo, como organismo certificador europeo que es, unos determinados criterios funcionales.

Para cerrar este apartado cabe decir que no existe un consenso pleno a nivel mundial sobre qué organización y estándar debería adoptarse. Las propuestas de mayor relevancia son el ISO 13606 del **C**omité **E**uropeo de **N**ormalización, la ***C****linical* ***D****ocument* ***A****rchitecture* de la organización norteamericana **HL7** (*Health Level* 7) y la propuesta de ***OpenEHR*** estrechamente ligada al ISO13606.

2. Contexto Médico

2.1. Introducción

La sanidad es uno de los entornos más complejos desde el punto de vista organizativo y tecnológico. La gran variedad de información que cambia constantemente junto con las reglas ético legales establecidas hacen que el compartir información clínica de forma efectiva, segura y eficiente sea una meta muy difícil de alcanzar. El intercambio de información clínica es una necesidad para el sector sanitario. Sin embargo, los datos de un paciente no suelen encontrarse alojados en un solo sistema de HCE. Esto conduce a la existencia de fuentes de datos heterogéneas creando una importante brecha entre el valor de la información que los HCE contienen y el que podrían tener. Se hace pues necesario el desarrollo de arquitecturas de HCE que permitan compartir la información clínica entre los distintos centros hospitalarios.

En los últimos años han surgido en Europa diversas iniciativas con el objetivo de avanzar en el aspecto técnico de la estandarización e interoperabilidad de las HCE. Algunas de estas iniciativas son GEHR, SYNAPSES, SYNEX y GALEN. A su vez, se ha instado a las organizaciones de desarrollo de estándares CEN, ANSI, ISO, CENELEC y ETSI a desarrollar un programa de estandarización acelerada en la informática de la salud.

2.2. Sistemas de Historia Clínica Electrónica

Como Stephanie L.Reel y Steven F.Mandell exponen en *Aspects of Electronic Health Record Systems* (1) *,* los sistemas de Historia Clínica Electrónica (HCE) han llevado a cabo una gran evolución en las últimas décadas. Los primeros sistemas tenían como objetivo trasladar la mayor cantidad posible de datos del paciente del papel a registros electrónicos. Ello implicó la captura y proceso de los datos, además de proporcionar soporte a funciones administrativas y de gestión.Como hasta la llegada de estos sistemas toda la información era mantenida en forma de documentos en papel, la llegada de esta tecnología fue muy prometedora ya que suponía un gran avance al reducir la redundancia y facilitar las tareas de tratamiento de la información.

Con el cambio hacia una sociedad con una mayor movilidad, el paradigma de atención sanitaria cambió de una atención centralizada a una descentralizada, donde la atención es llevada a cabo por diferentes proveedores que pueden estar emplazados en lugares diferentes. Lo anterior propició que las tecnologías de la información debieran aportar soluciones para la gestión de la salud en el nuevo paradigma. Con ello, los sistemas de HCE fueron incorporando posibilidades

de captura de más tipos de datos, extracción de fuentes heterogéneas y almacenamiento electrónico de datos. Actualmente la información se organiza y almacena para permitir un uso eficiente, seguro y de calidad que permita dar un buen servicio de salud. Sin embargo en la actualidad encontramos que los sistemas de HCE han acabado igual de fragmentados que en los tiempos en los que la información se almacenaba en papel. Esto se debe a que muchos de los archivos permanecen en forma de documentos escritos y que muchos grupos de proveedores han creado sus propios "silos electrónicos".

La definición más reciente de HCE la provee el *Institute of Medicine* (IOM) de EEUU. Según ella el sistema debe incluir:

- Colección longitudinal de información sanitaria acerca de personas. Donde la información sanitaria se define como información relativa a la salud de un individuo .
- Acceso electrónico inmediato a la información de individuos o poblaciones solo por sujetos autorizados.
- Mejorar la calidad, seguridad y eficiencia del cuidado del paciente proporcionando conocimiento y apoyo a la toma de decisiones.
- Apoyar de forma eficiente a los procesos involucrados en la atención sanitaria.

El IOM propone cinco criterios como guía para determinar las funcionalidades de las HCE. Estos contemplan la mejora de la seguridad de pacientes, el soporte a los procesos de la atención sanitaria, facilitar la gestión de estados crónicos, mejorar la eficiencia y determinar si es factible implementar el sistema. Además el IOM (*Institute Of Medicine*) ha identificado las funcionalidades del núcleo de un sistema de HCE como:

- Información y datos relativos a salud.
- Pedidos de entrada/gestión.
- Soporte a decisiones.
- Gestión de resultados.
- Comunicaciones y conectividad electrónica.
- Apoyo a pacientes.
- Procesos administrativos.
- Informe y demografía de datos de salud.

El IOM propone la construcción de la HCE de forma incremental utilizando los sistemas de información clínicos como los bloques que se usarán para construirlo. Además los cuatro escenarios donde debe situarse el sistema de HCE son hospital, atención primaria, atención domiciliaria y residencias. Para ello el IOM provee de un marco de trabajo a través del cual puedan ser desarrolladas las estructuras y sistemas que permitan alcanzar los retos que surjan durante la vida útil de un sistema de HCE que alcance a los diferentes proveedores de salud.

2.2. Concepto de interoperabilidad semántica

Según Wikipedia la interoperabilidad semántica (IopS) se define como:

Habilidad para interpretar de forma automática la información intercambiada teniendo en cuenta su significado con el fin de producir respuestas útiles a los dos sistemas en comunicación. Para conseguir interoperar semánticamente, ambos extremos de la comunicación deben converger en un modelo de referencia común. A su vez, el contenido de la información intercambiada debe estar definido sin ambigüedades. Esto significa que lo que se envía es lo mismo que lo que se entiende.

Actualmente se definen una serie de niveles de interoperabilidad que deben ser alcanzados gradualmente para llegar a la plena consecución de esta. Estos niveles son (2):

- ***Nivel 0***: No existe ningún tipo de interoperabilidad. No hay ningún tipo de acceso a los datos del paciente entre los distintos centros que lo han atendido.

- ***Nivel 1***: interoperabilidad técnica y sintáctica. Un centro puede recibir documentos de otro sobre un determinado paciente por medios como pueda ser vía web, ftp, mail etc. No hay ningún tipo de codificación ni estructuración de la información compartida.

- ***Nivel 2***: comprende 2 subniveles de interoperabilidad parcial: Un centro puede recibir de otros extractos de la HCE del paciente. Estos extractos, aunque puede que incompletos, contienen información sensible acerca del paciente como alergias, demografía, diagnósticos etc. Además la información accedida está codificada usando esquemas de codificación internacionales de manera que los datos pueden ser decodificados e interpretados de forma inmediata por el facultativo interesado.

  Nivel 2a: interoperabilidad semántica unidireccional de algunos fragmentos significativos de la HCE.

  Nivel 2b: interoperabilidad semántica bidireccional de algunos fragmentos significativos de la HCE.

- ***Nivel 3***: integridad semántica completa. Es posible acceder y compartir la información clínica de una forma transparente y sin barreras. Es posible interpretar los datos obtenidos. El acceso a la información clínica se hace sin problemas de idioma, heterogeneidad de sistemas o tecnologías. El único requisito es la autentificación contra el sistema de forma que no queden comprometidos los datos del paciente.

La interoperabilidad semántica persigue objetivos como facilitar la codificación, transmisión y uso de servicios de salud de una forma fluida y sin barreras. Desde el punto de vista geográfico podemos clasificar la interoperabilidad en: local, regional, nacional e internacional.

La interoperabilidad semántica afecta a distintas áreas de la información clínica:

- IopS para los datos de pacientes individuales: implica el acceso a datos de diagnóstico, monitorización, alertas, entorno del individuo etc.

- IopS para datos de una determinada población: implica el acceso a informes, datos económicos, supervisión, epidemiología etc.

Además, al manejar la información los sistemas implicados en conseguir la interoperabilidad semántica deberán tener en cuenta las siguientes características a la hora de gestionar los datos clínicos:

- ***Consistencia***: el sistema que recibe la información debe ser capaz de reconocer lo que se le está enviando. Esto lleva al uso de identificadores unívocos entre todos los sistemas que sean susceptibles de ser semánticamente interoperables.

- ***Comprensibilidad***: Evitar la confusión a la que puede llevar la gran cantidad de información agregada. Al agregar datos de información poblacional la densidad de estos puede abrumar al receptor humano confundiéndolo. Se hace esencial la adecuada presentación de la información clínica por parte del sistema.

- ***Reproducibilidad***: Los individuos encargados de recopilar y codificar los datos deben ser de una alta fiabilidad. De esta manera un clínico que acceda a la información de un paciente en otra nación debe tener una absoluta seguridad de que los datos accedidos son fiables. Además el extracto accedido debe ser idéntico al generado en otro instante o lugar.

La IopS es un objetivo que debe considerarse a medio y largo plazo para asegurar su éxito. No existe una fórmula mágica que permita instalar una aplicación que convierta a nuestro sistema en plenamente interoperable con tan solo hacer clic en su ventana de instalación. Lograr ser interoperables semánticamente es un arduo proceso que no se puede lograr sin la implicación de todos los estratos que toman parte en la prestación de servicios médicos. Desde los facultativos, pasando por la dirección de los centros y los gobiernos regionales y nacionales.

Alcanzar la interoperabilidad semántica de los sistemas de información de salud requiere de una serie de herramientas, tecnologías y acuerdos mínimos que la hagan viable. Son tres los

pilares básicos sobre los que se asienta la interoperabilidad semántica de la información: un estándar o modelo de referencia para representar la información, terminologías compartidas que definan el vocabulario utilizado para describir los datos y definiciones formales de conceptos clínicos (arquetipos) que sean capaces de aunar las dos partes anteriores. Los arquetipos pueden considerarse como una primera capa de descripción semántica de la información, aunque aún requieren algo más para alcanzar el grado de interoperabilidad semántica deseado (3).

2.3. Modelo dual y arquetipos

La sanidad es un dominio muy variable con el tiempo. Esto provoca que el mantenimiento de los sistemas de información clínica se haga extremadamente complicado hasta alcanzar el punto de obsolescencia mucho más rápidamente que una aplicación normal. La razón de este costoso mantenimiento es, en gran parte debida, a la gran cantidad de restricciones que recaen sobre la implementación de la aplicación y a la continua evolución de la práctica clínica donde los cambios en la información no son la excepción sino la norma.

A continuación se aprecia la curva real del software que plantea Roger S. Pressman (4). En ella se ve como el software va poco a poco deteriorándose a lo largo del tiempo debido a los fallos que introduce sobre él el mantenimiento. Estos fallos acaban por hacer obsoleto al software y precisar de su sustitución.

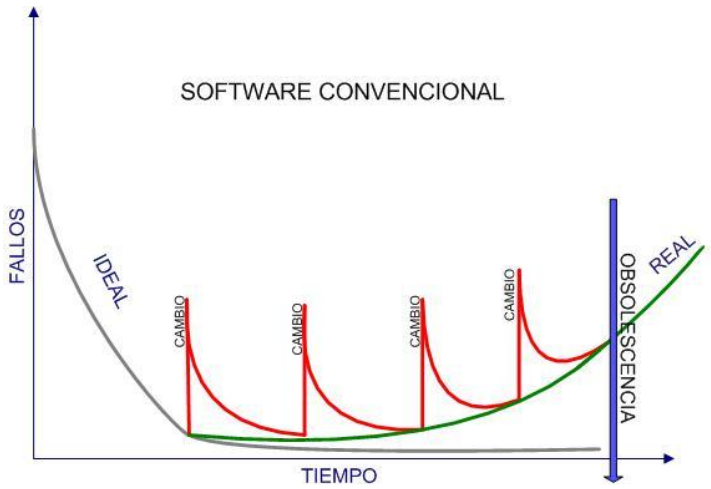


Figura 1: Curva de software convencional.

Adaptando esa misma gráfica a las aplicaciones de las TIC (Tecnologías de Información y las Comunicaciones) de la salud vemos que su vida útil es mucho más corta debido a la complejidad e intensidad del mantenimiento que necesitan. Así pues, aunque un modelo tradicional implica un proceso de desarrollo más rápido y fácil, en el campo de la salud, queda

en desuso mucho antes por el coste de mantenimiento. Se hacen así necesarios sistemas que permitan una adaptación al cambio y una facilidad de mantenimiento sin precedentes. Para esto se han desarrollado nuevos modelos de software como el *Modelo Dual*.

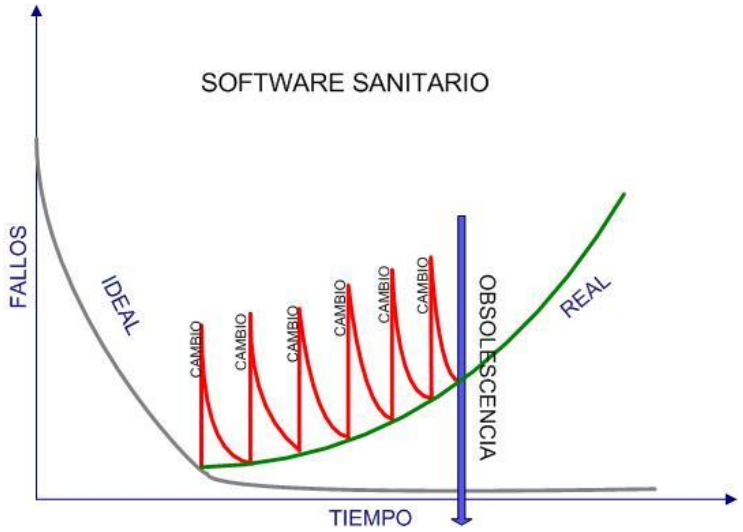


Figura 2: Curva software sanitario.

2.3.1. Modelo Dual

Los sistemas de información actuales contemplan en su modelo de datos tanto las restricciones de la información como las más particulares del dominio específico que tratan. Esto hace que el modelo subyacente al sistema sea una maraña que enlaza entidades propias de los estándares con relaciones particulares del conocimiento del dominio. Para dar solución a esta problemática se desarrolló el dominado Modelo Dual (5) nacido a partir del proyecto europeo GEHR (*Good Electronic Health Record*).

La propuesta del modelo dual se basa principalmente es la separación de información y conocimiento. El nivel de información define las entidades básicas y estables del dominio a partir de las cuales se construye, mediante la definición de restricciones y relaciones, la capa de conocimiento.

Nivel de información:

Este nivel está compuesto por un pequeño modelo de datos, denominado modelo de referencia

(MR), que contiene los conceptos no volátiles del sistema de información. Un ejemplo sería la parte 1 de la norma CEN EN13606.

La metodología dual se asemeja bastante a los postulados propuestos por la ingeniería del dominio ámbito donde existe un interés creciente dentro del campo de la ingeniería del software. Por ingeniería del dominio se entiende el obtener una descripción clara de un dominio particular con el fin último de definir estructuras de datos reutilizables que faciliten el posterior desarrollo de nuevos sistemas de información. Los conceptos de negocio del modelo de referencia son ejemplos claros de estas estructuras reutilizables

Tomando el ejemplo que expone Thomas Beale en su artículo *Archetypes: Constraint-based Domain Models for Future-proof Information Systems* (5) en el sistema de demográficos de un determinado hospital encontraríamos las siguientes entidades en un modelo de referencia: Personas, instalaciones hospitalarias, profesionales de la salud, agente de salud, miembro de la plantilla, administrador, paciente etc. En el ámbito de la historia clínica electrónica un modelo de referencia debe contener básicamente:

- Un conjunto de conceptos de negocio, cada uno de los cuales representa un tipo de bloque constitutivo de la historia clínicas con diversa granularidad.
- Los tipos de relaciones posibles entre los conceptos de negocio (herencia, agregación, composición, etc.).
- Estructuras de datos como valores simples, listas, tablas, árboles, series temporales, etc.
- Un conjunto de tipos de datos básicos como texto, término codificado, multimedia, cantidad física, rango, etc.

Nivel de Arquetipos:

La generalidad del MR se complementa con la especificidad de los arquetipos, el segundo nivel. Los arquetipos son definiciones formales de conceptos clínicos (por ejemplo, paciente, medida de la glucosa, informe de alta, etc.) que se definen por medio de restricciones sobre las entidades del MR. En esta metodología sólo el estable y simple MR se codifica en la base de datos, mientras que los numerosos y volátiles conceptos del dominio (arquetipos) se modelan por especialistas del dominio de manera separada. Ya que el software únicamente es dependiente del MR la incorporación de nuevos conceptos no requiere su modificación.

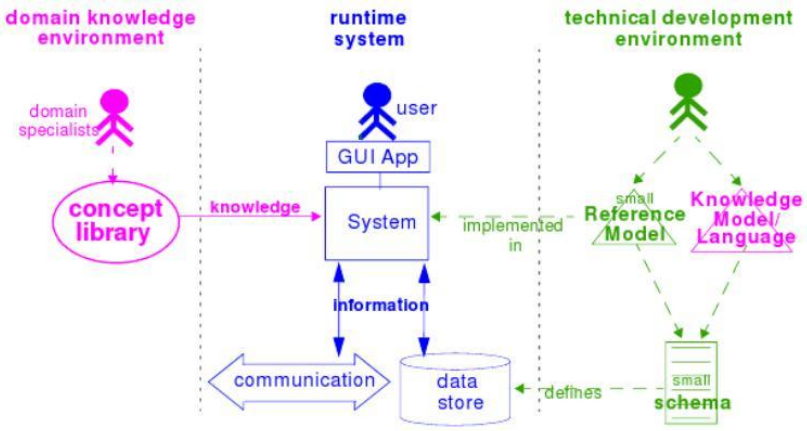


Figura 3: Modelo Dual de Thomas Beale 2002.

El siguiente ejemplo (6) muestra de forma más clara el concepto de modelo dual:

Luis Vila el día 20/08/2002 tenía una cantidad de Fibrinógeno en sangre de 256 mg/dl, esta información se refiere a Luis Vila y no a otra persona. Mientras que conocimiento se define como los datos y hechos que son ciertos para todas las instancias de las entidades a las que se refieren. Por ejemplo, el fibrinógeno es una proteína sintetizada por el hígado que se utiliza en el proceso de coagulación de la sangre, su cantidad se mide en miligramos por decilitro y el intervalo de referencia va desde los 200 mg/dl a los 450 mg/dl, esta información se puede aplicar a todas las personas.

Una de las consecuencias más interesantes del modelo dual es la separación que se obtiene entre la implementación del sistema informático y la definición de los conceptos del dominio. Más concretamente, el software sigue siendo una implementación del modelo de referencia y los datos gestionados por el sistema son instancias del modelo de referencia, pero los conceptos se pueden definir por especialistas del dominio cuando el sistema ya está implantado. Como consecuencia los sistemas desarrollados tienen el potencial de poder ser implantados antes incluso de la creación de los conceptos del dominio, es decir, su creación es independiente del desarrollo del software y, por tanto, pueden ser particulares de cada sistema u organización.

2.3.2. Especificación de Arquetipos

El arquetipo es un modelo formal que define un concepto del dominio por medio de restricciones sobre las estructuras de datos contenidas en el modelo de referencia subyacente.

Las restricciones podemos verlas como especificaciones que dictan las características que deben satisfacer las instancias de un modelo de referencia para que sean instancia de un concepto de negocio válido para un dominio en particular. Los tipos de restricciones básicas son:

- Restricciones sobre el dominio de los atributos como declaración del valor máximo y/o valor mínimo o la enumeración de los valores aceptados.
- Restricciones sobre las relaciones de agregación entre arquetipos. Esto incluye tanto la especificación de las condiciones que debe cumplir un arquetipo para que pueda estar contenido en otro como la cardinalidad de esta relación, es decir, cuántas instancias del arquetipo contenido pueden aparecer dentro de una instancia del arquetipo contenedor.
- Restricciones sobre la obligatoriedad de los atributos y número de ocurrencias posibles si éstos son multivaluados.

Así, mediante los arquetipos se nos dota de una capacidad de expresividad mucho más rica y completa para especificar con mayor detalle la semántica de la información clínica a compartir.

ESTRUCTURA

Existe un lenguaje formal para la definición de arquetipos denominado ADL (*Archetype Description Language*) desarrollado por el consorcio OpenEHR y que ha sido adoptado por EN13606. ADL posee una sintaxis sencilla que se apoya en otras dos sintaxis para expresar las restricciones y los datos respectivamente. cADL es la sintaxis usada para expresar la estructura de restricciones, mientras que dADL es la sintaxis utilizada para expresar metainformación como lenguaje, descripción, ontología e histórico de revisiones.

La definición del arquetipo se estructura básicamente en tres secciones principales: cabecera, definición y ontología. Las Figuras 4 y 5 muestran con mayor detalle la estructura de un arquetipo en ADL.

cabecera: En la cabecera de la definición del arquetipo se muestra la información referente a este como puede ser el identificador, idiomas disponibles, información autorizada etc.

definición: En la figura vemos las secciones en las que típicamente está estructurado un arquetipo.

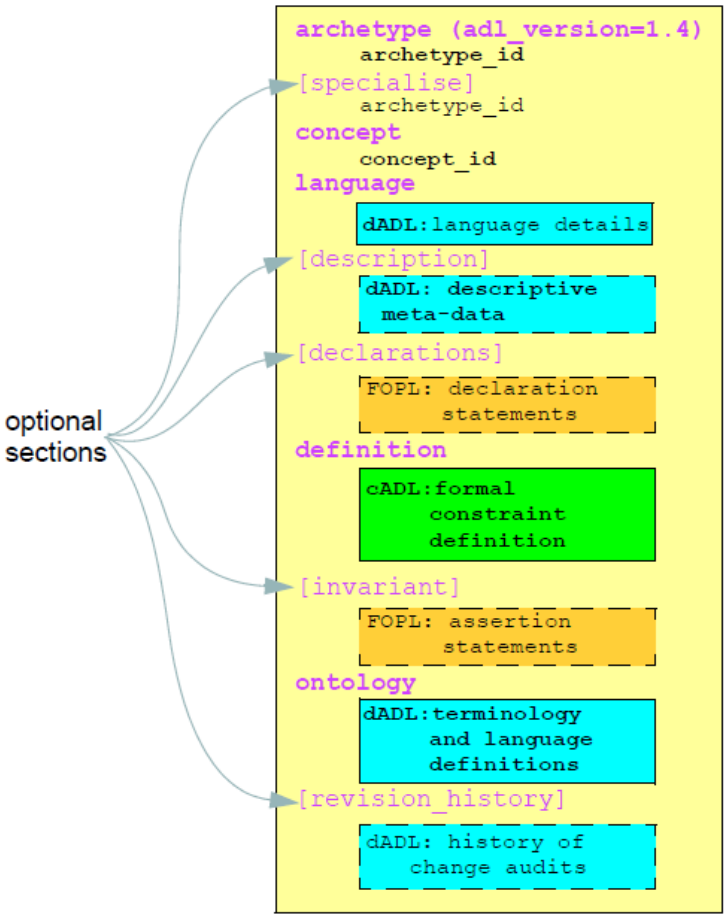


Figura 4: Estructura del arquetipo.

A efectos del trabajo desarrollado como proyecto el apartado de más interés sería el de *definition*. En él se lleva a cabo la descripción de restricciones formales sobre el arquetipo. Este tipo de descripciones se tratan con más detalle en el próximo apartado.

La definición de arquetipo se estructura en las siguientes secciones:

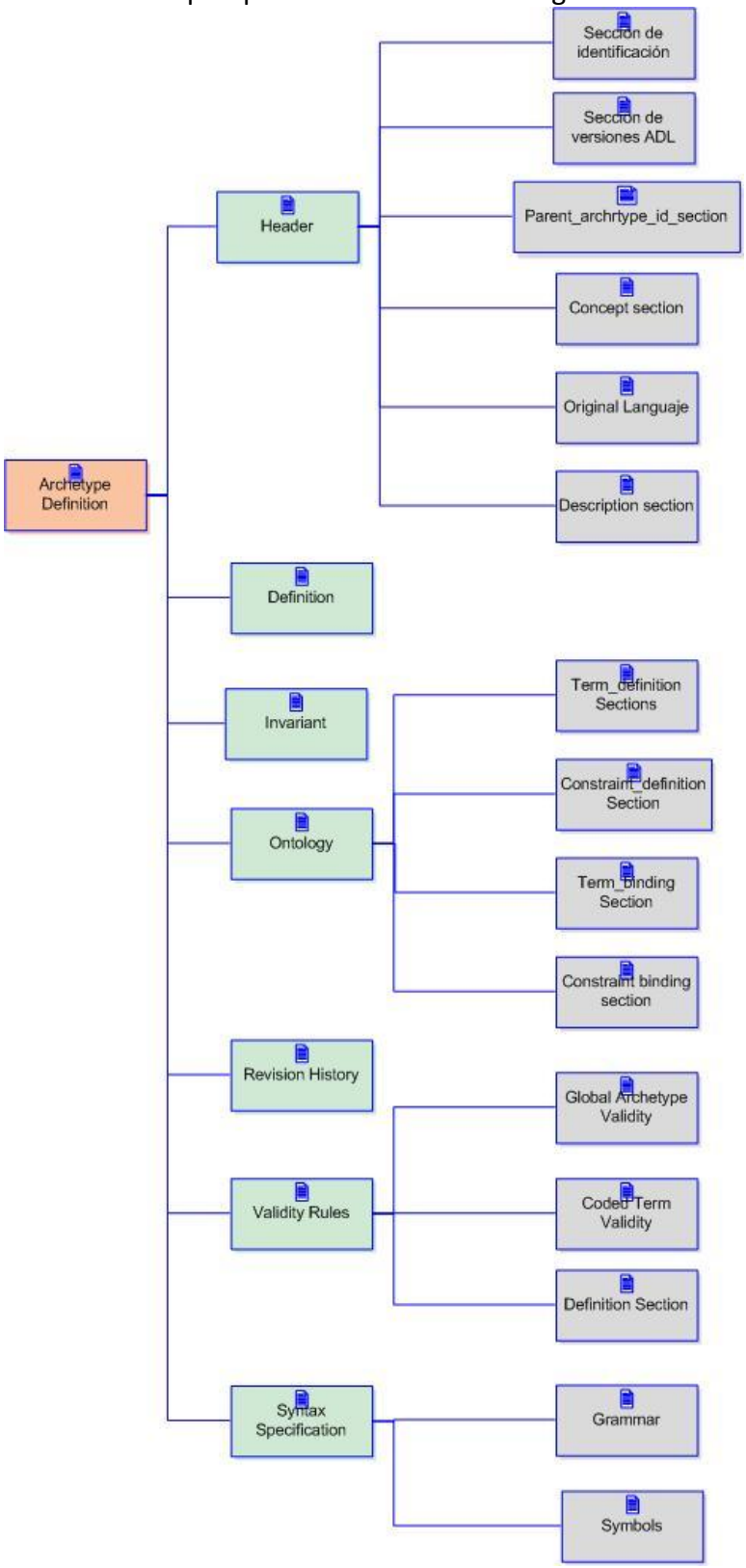


Figura 5: Secciones del arquetipo.

Ejemplo:

En el siguiente ejemplo extraído del manual adl1.4 de OpenEHR (7) vemos la definición del concepto guitarra con ADL. Podemos observar que una guitarra es una especialización del concepto de negocio *INSTRUMENT*. Se define por medido de restricciones sobre tres atributos: *size*, *date_of_manufacture* y *parts*.

```
archetype (adl_version=1.4)
    adl-test-instrument.guitar.draft
concept
    [at0000] -- guitar
language
    original_language = <[iso_639-1::en]>
definition
    INSTRUMENT[at0000] matches {

        size matches {|60..120|}                    -- size in cm
        date_of_manufacture matches {yyyy-mm-??}    -- year & month ok
        parts cardinality matches {0..*} matches {
            PART[at0001] matches {                  -- neck
                material matches {[local::at0003,   -- timber
                                   at0004]}         -- timber or nickel alloy
            }
            PART[at0002] matches {                  -- body
                material matches {[local::at0003}   -- timber
            }
        }
    }
ontology
    term_definitions = <
        ["en"] = <
            items = <
                ["at0000"] = <
                    text = <"guitar">;
                    description = <"stringed instrument">
                >
                ["at0001"] = <
                    text = <"neck">;
                    description = <"neck of guitar">
                >
                ["at0002"] = <
                    text = <"body">;
                    description = <"body of guitar">
                >
                ["at0003"] = <
                    text = <"timber">;
                    description = <"straight, seasoned timber">
                >
                ["at0004"] = <
                    text = <"nickel alloy">;
                    description = <"frets">
                >
            >
        >
    >
```

Figura 6: Especificación del arquetipo guitarra con ADL.

2.3.3. Tipos de restricciones en arquetipos

Este apartado extraído de la tesis de J. Alberto Maldonado (6) se nos muestran las restricciones que es posible describir sobre un arquetipo. En él se tratan en primer lugar las restricciones dADL, de modelado de datos, y en segundo lugar, las restricciones cADL, que permiten establecer restricciones sobre estructuras de datos.

Restricciones dADL:

dADL es una sintaxis muy simple para expresar datos, ya sean éstos instancias de un modelo de referencia (extracto de historia clínica en nuestro caso) o metainformación contenida en la definición de los arquetipos. El modelo de datos subyacente es puramente jerárquico, las estructuras de datos se construyen a partir de la plantilla general:

Identificador_atributo = <valor>

A cada valor se le asigna un identificador de atributo (etiqueta), que debe provenir del modelo de referencia cuando se utiliza dADL para expresar instancias. Un valor puede estar compuesto por una secuencia de longitud arbitraria de otras estructuras, esto permite crear estructuras anidadas de profundidad arbitraria. Los nodos internos de los árboles de datos están etiquetados con identificadores de atributos y las hojas están etiquetadas con un valor perteneciente a alguno de los tipos primitivos: cadena de caracteres (string), enteros, reales, booleanos, caracteres y fechas y horas. En dADL no es necesario indicar el tipo de un atributo cuando éste es primitivo, basta con especificar el valor, de esta forma se consigue mayor independencia con respecto a los nombres y estructura de los tipos primitivos utilizados en el modelo de referencia.

Un identificador de atributo puede ir acompañado por un cualificador, que debe ser instancia de cualquier tipo básico comparable, por ejemplo: dirección (“domicilio”) o dirección (“trabajo”). Los atributos cualificados permiten representar listas o tablas de valores independientemente de la forma en que estas estructuras se definen en los sistemas origen y destino de la información. dADL permite, también, asociar un identificador a las instancias de un tipo por medio de la estructura:

TIPO_A [id_1] = < Identificador_atributo = <$valor_1$>

…
Identificador_atributo = <$valor_n$>

>

Donde construimos una instancia del tipo “TIPO_A” y además le asociamos un identificador “id_1” que permitirá hacer referencia a esta instancia en otro lugar y por tanto reutilizar datos.

Restricciones cADL:

cADL es una sintaxis que permite definir restricciones sobre estructuras de datos, cuyas instancias son conformes a un modelo de referencia expresado en un modelo orientado a objetos. Tres normas generales rigen la definición de estas restricciones:

Los identificadores de tipos, relaciones y atributos usados en un fragmento de cADL deben corresponderse con algún identificador de tipo, relación o atributo del modelo de referencia.

Solo es necesario declarar restricciones para aquellas partes del modelo de referencia que necesitan ser restringidas, por tanto, pueden existir más identificadores de atributos, tipos y relaciones en el modelo de referencia que los utilizados en los arquetipos.

Las restricciones definidas en un arquetipo no pueden ser más fuertes que las existentes en el modelo de referencia, así por ejemplo, un atributo obligatorio en el modelo de referencia no puede ser opcional en un arquetipo.

Las restricciones en cADL se definen por medio de cláusulas que restringen las instancias de los tipos y los valores de los atributos. Las restricciones se estructuran en bloques anidados que comienzan por un nombre de tipo (clase) o por un nombre de atributo del modelo de referencia. La estructura general de una restricción en cADL es la siguiente:

entidad {Restricción}

donde entidad es un nombre de tipo o de atributo.

En cADL el anidamiento de nivel superior está compuesto por un bloque que contiene restricciones sobre tipos, seguido por restricciones sobre atributos (del tipo del nivel superior), seguido por restricciones sobre tipos (los tipos debe ser del atributo restringido por el bloque de nivel superior) y así sucesivamente. Se utilizan los términos bloque de objeto o nodo de objeto para referirse a un bloque encabezado por un nombre de tipo, mientras que un bloque introducido por un nombre de atributo se denomina bloque de atributo o nodo de atributo.

En la figura se puede encontrar un ejemplo de cADL donde se define una restricción sobre las instancias del tipo (clase) PERSONA. Todo lo que está incluido dentro del bloque de PERSONA constituye la restricción. Los dos bloques del primer nivel de anidamiento definen restricciones sobre dos atributos PERSONA, en concreto nombre y dirección. Cada uno de estos bloques contiene a su vez otros que restringen los tipos de los atributos y así sucesivamente.

Veamos con detalle los tipos de restricciones posibles en cADL:

Existencia. Esta restricción solamente se aplica a los atributos, por tanto, solo aparece en los bloque de objeto. Permite especificar si debe existir en las instancias de datos un valor para el atributo en cuestión. La restricción por defecto es {1..1} que denota obligatoriedad. Por ejemplo:

ENTRY [at001] ∈ {act_id existence ∈{0..1}}

indica el valor del atributo act_id de la clase ENTRY (definida en el modelo de referencia de EN13606) no es obligatorio.

Cardinalidad. Mientras la existencia expresa la obligatoriedad o no de un valor para un atributo, con la cardinalidad indicamos si el tipo del atributo es un contendor, como una lista o un conjunto, así como el número de elementos que puede contener. El siguiente ejemplo expresa que el atributo *annotations* de la clase *ENTRY* no es obligatorio y que es un contenedor que puede tener cero o muchas instancias la clase *CS_ANNOTATION*.

ENTRY [at001] ∈ {

annotations existence ∈ {0..1} cardinality ∈ {0..*} ∈ { CS_ANNOTATION [at002] ∈ {…}

}

Ocurrencia. Se aplica únicamente a clases contenidas en el modelo de referencia, por tanto solo aparecen en nodos de objeto. Permite indicar el número de instancias que pueden aparecer de la clase en cuestión, la restricción por defecto es {1..1}. El siguiente ejemplo describe restricciones sobre instancias de grupos de personas (asociaciones o clubs), que deben estar compuestos por más de un miembro.

```
GRUPO [at101] ∈ {
        Tipo ∈ {asociación, club}
        Miembros cardinality ∈ {*} {
                PERSONA [at103] occurences {1..*} ∈ {
                        Cargo ∈ {"miembro"}
                        ....
                }
        }
}
```

Sobre tipos primitivos. Limitan el dominio de estos tipos.

- Strings. A través de una expresión regular o indicando una lista de valores posibles.
- Enteros. Por medio de un intervalo, lista de enteros o un único valor.
- Reales. Igual que los enteros.

- Booleanos. Cualquier subconjunto de {True, False}
- Caracteres. Indicando un único valor o una expresión regular.
- Fechas, horas y duraciones. Por medio de una plantilla de ISO8601. Además para las fechas y horas es posible indicar un valor concreto o un intervalo.

Podemos omitir el nombre de los tipos primitivos, por tanto atributo ∈ { TIPO_ATOMICO ∈ {Restricción}} es equivalente a atributo ∈ {Restricción} .

Alternativas. En cADL pueden aparecer bloques consecutivos que restringen objetos de la misma clase, que pueden tener dos significados posibles dependiendo de la combinación de las ocurrencias individuales de cada bloque y la cardinalidad del atributo contenedor:

- En el caso que el atributo contenedor tenga una cardinalidad de {1..1} cada uno de los bloques se considera una alternativa y las instancias solamente tienen que ser conformes a una de ellas (en el caso de cardinalidad {0..1} las instancias tienen que ser conformes a una o a ninguna de las alternativas). En el siguiente ejemplo se define instancias de datos que contienen velocidades máximas de diversos países que pueden estar medidas en kilómetros por hora (km/h) o en millas por hora (mph).

```
ELEMENT[04]   ∈ {          --velocidad máxima
 país ∈ {/.+/}
 valor ∈ cardinality ∈{1..1} {
          Cantidad[at01] ∈ ocurrences ∈{0..1} {
                    magnitud ∈ {0..*}
                    propiedad ∈ {velocidad}
                    unidades ∈{"km/h"}
                      }
          Cantidad[at02] ∈ ocurrences ∈ {0..1} {
                    magnitud ∈ {0..*}
                    propiedad ∈ {velocidad}
                    unidades ∈{"mph"}
                      }
          }
}
```

- El atributo contenedor es multivaluado entonces cada uno de los valores individuales puede ser instancia de cualquiera de las alternativas. En el siguiente extracto se detalla que un grupo está compuesto por múltiples persona una de las cuales debe ser el presiente y el resto miembros.

```
GRUPO [at101] ∈ {
                    Tipo ∈ {asociación, club}
                    Miembros cardinality ∈ {1..*} {
                              PERSONA [at102] occurences {1} ∈ {
                                        Cargo ∈ {"presidente"}
                                        ....
                              }
                      PERSONA [at103] occurences {0..*} ∈ {
```

```
                                        Cargo ∈ {"miembro"}
                                        ....
                                    }
                            }
                    }
```

Restricción "any". La ausencia de restricción se expresa por medio de la restricción "any", que se denota por medio de un asterisco entre paréntesis. Esta restricción es útil para indicar que alguna propiedad puede tener cualquier valor de los permitidos por el modelo de referencia o para expresar que una propiedad es de un tipo determinado pero puede tener cualquier valor internamente.

Referencias internas. En cADL es posible hacer referencias a restricciones anteriores declaradas en otros bloques, esto se consigue con la palabra reservada use_node siguiendo el siguiente patrón:

Use_node Tipo_de_objeto_referenciado Camino_del_objeto

Referencias a arquetipos. Con cADL es posible definir relaciones de composición entre arquetipos. En cualquier punto de una definición en cADL es posible incluir una restricción que permite utilizar arquetipos ya definidos en vez de volver a especificar las restricciones deseadas. A estas restricciones se denominan archetype slot. Un archetype slot se introduce con la palabra clave allow_archetype y contiene un conjunto de sentencias en cADL que define el conjunto de arquetipos que pueden aparecer en ese punto. En ocasiones, estas sentencias, pueden hacer referencia a arquetipos específicos, pero en general definen un conjunto más o menos amplio de arquetipos.

Invariantes. Las restricciones que involucren a más de una propiedad (atributo o relación) no pueden ser expresadas por medio de la sintaxis presentada. En cADL este tipo de restricciones se denominan invariantes y se definen por medio de fórmulas de la lógica de predicados de primer orden cuyo valor de verdad se pueda computar. Los invariantes deben aparecer dentro de una sección identificada por la palabra clave invariant. Además, deben estar situados dentro del bloque más interno que contiene a todas las propiedades referenciadas por el invariante. El siguiente ejemplo, establece que si en los datos existe una instancia de un tipo dirección entonces ésta debe existir una instancia del tipo nombre:

Invariant

Validez: exists dirección implies exists nombre

Que es equivalente a la siguiente fórmula expresada en una sintaxis más estándar:

$\forall x \; \exists y \;$ (dirección(x) $\rightarrow$ nombre(y))

Caminos.cADL incorpora una sintaxis para especificar referencias a nodos ya sean éstos de objeto o de atributo. Los caminos alternan nombres de objetos y de atributos siguiendo la estructura jerárquica del cADL y siguen el patrón TIPO/atributo/TIPO/atributo… . Un camino acaba con un nombre de atributo o con un nombre de atributo seguido por un identificador de nodo de objeto encerrado entre paréntesis cuadrados.

Por ejemplo, el siguiente camino referencia al nodo de objeto con identificador [at0004] incluido en el nodo asociado al atributo nombre desde el nodo raíz con identificador [at0001]:

/[at0001]/nombre[at0004]/

2.3.4. Modelo de objetos de arquetipos

El modelo de objetos de arquetipos es un modelo orientado a objetos genérico para la representación de arquetipos. Puede ser utilizado como base para el desarrollo de software pare el procesado de arquetipos, independientemente del formato utilizado para su persistencia, por ejemplo ADL. El modelo de arquetipos es la base del sistema LinkEHR-Ed, ya que permite trabajar con arquetipos basados en cualquier modelo de referencia. Como se observa en la figura, las clases del modelo de objetos de arquetipo permiten la definición de cada una de las entidades (clases y atributos) que pueda tener un determinado modelo de referencia. Así, en el modelo de objetos encontramos entidades para representar objetos complejos (clases), atributos de tipo simple (atributos atómicos) y compuesto (atributos no atómicos), referencias a objetos, cardinalidades y otros.

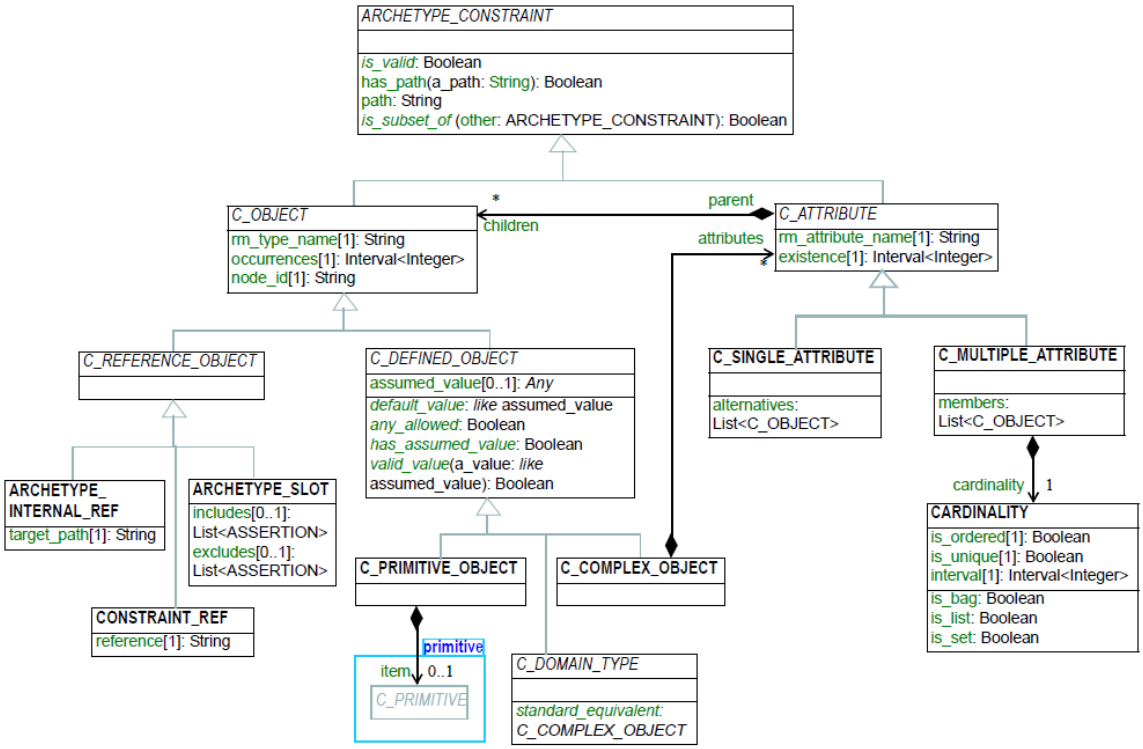


Figura 7: Modelo de arquetipos.

3. Editor gráfico de LinkEHR-Ed

3.1. Propósito y funcionamiento

El propósito general para el que fueron pensadas las arquitecturas basadas en el modelo dual fue la del desarrollo de nuevos sistemas de información sanitarios. Pero la mayoría de las organizaciones sanitarias ya disponen de sistemas informáticos que manejan datos generalmente con estructura propia y no normalizada. Por tanto, surge la necesidad de adaptar dichos sistemas para que sean compatibles con algún estándar de HCE, como puede ser el definido por el EN13606, de tal manera que la información clínica que contienen pueda ser compartida con otros sistemas. Básicamente LinkEHR-Ed explora el uso de arquetipos como mecanismo para conseguir la estandarización e integración semántica de información clínica ya existente. Los objetivos principales son dos. Por un lado utilizar los arquetipos como descriptores de los repositorios de datos a integrar y al mismo tiempo ocultar su heterogeneidad. Es decir, utilizarlos como una capa semántica sobre las fuentes de datos que asocia a los datos almacenados en éstas una semántica clínica específica. El segundo objetivo es utilizar los arquetipos para hacer pública la información clínica ya existente y no normalizada en documentos XML conformes a la norma (modelo de referencia) elegida. Para ello, los arquetipos se consideran como esquemas que definen la estructura y contenido de las instancias válidas de la norma seleccionada. Los usuarios extraen información por medio de la instanciación de uno o varios arquetipos. Por tanto, los arquetipos definen los conceptos clínicos que se comparten y los extractos de historias clínicas que se facilitan a las aplicaciones clientes son siempre instancias de arquetipos. Con esto se consigue reducir el problema de conocer el contenido y organización de múltiples fuentes de información al problema de conocer el contenido de un conjunto de descripciones formales de conceptos clínicos (arquetipos) que un usuario familiarizado con el dominio clínico puede conocer o entender fácilmente e incluso pueden ser “entendidos” por un ordenador.

LinkEHR-Ed (http://pangea.upv.es/linkehr) es una herramienta visual implementada en Java bajo la plataforma Eclipse[1] que permite:

- la edición de arquetipos, los cuales pueden estar basados en múltiples modelos de referencias
- la especificación de correspondencias (*mappings*) entre los arquetipos y las fuentes de datos
- la generación semiautomática de scripts de transformación de datos expresados en XQuery, los cuales transforman datos no normalizados en documentos XML conformes

[1] www.eclipse.org

con el modelo de referencia (previsiblemente una norma o estándar) y que a su vez satisfacen las restricciones sobre los datos impuestas por los arquetipos.

Se utilizará el término arquetipo de integración para nombrar un arquetipo para el cual se ha definido una correspondencia (*mapping*) con un conjunto de fuentes de datos, es decir:

Arquetipo de integración = arquetipo + correspondencia.

Existe una relación de uno a muchos entre los arquetipos y los arquetipos de integración. Dado un arquetipo, puede existir diversas correspondencias, al menos una por cada una de las fuentes de datos que queremos estandarizar y describir su contenido. La situación más común es aquella donde el arquetipo es compartido por diversas organizaciones (es decir, la definición de conceptos clínicos) pero cada organización posee sistemas propietarios de tal forma que es necesario definir correspondencias distintas. Estas correspondencias son propias de cada organización y no tiene sentido compartirlas. LinkEHR-Ed es una herramienta para la definición de arquetipos de integración (8).

La figura 8 describe el proceso completo de edición de un arquetipo de integración. El proceso comienza con la importación de un MR descrito por medio de un XML Schema. LinkEHR-Ed puede trabajar con múltiples modelos de referencia, característica que la hace única en la actualidad. Obviamente este paso únicamente es necesario realizarlo una única vez por cada MR. Una vez importado, es posible definir arquetipos basados en él o cargar arquetipos ya existentes, como los disponibles en un repositorio público de arquetipos. A continuación los usuarios deben definir las correspondencias con las fuentes de datos y a partir de éstos la herramienta genera un script XQuery que transforma los datos en un documento XML conforme con el MR y que a la vez satisface todas las restricciones impuestas por el arquetipo.

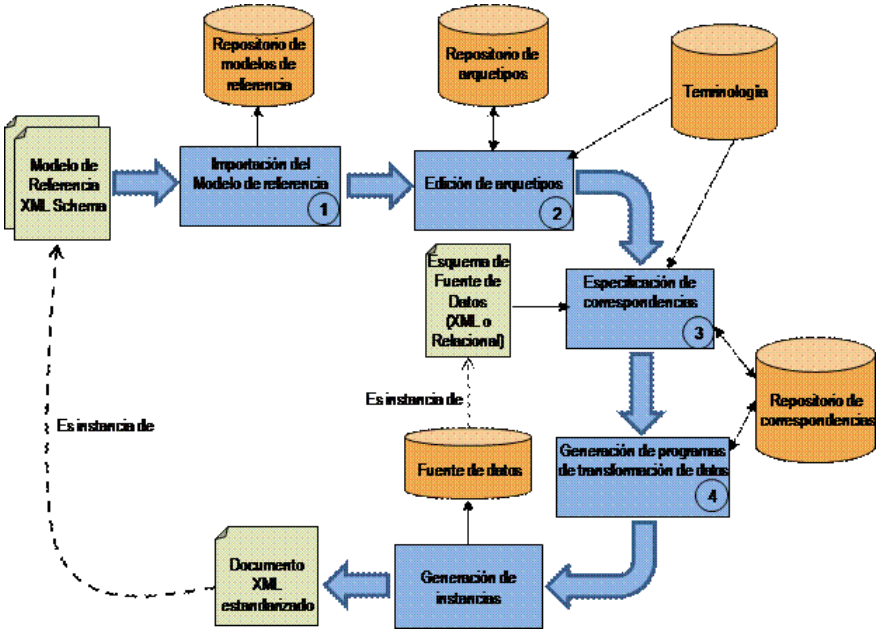


Figura 8: Proceso de estandarización LinkEHR-Ed.

3.2. Edición de Arquetipos en LinkEHR-Ed

En este apartado se describirá brevemente el entorno gráfico de edición de arquetipos de LinkEHR-Ed, sobre el cual se apoya el módulo de definición de correspondencias objeto de este proyecto (9). La edición se apoya en el modelo de objetos de arquetipos. Este modelo de arquetipos se utiliza para representar todos los modelos de referencias. Cuando un nuevo modelo de referencia se incorpora a la herramienta, el usuario debe indicar el conjunto de clases que pueden ser utilizadas para definir arquetipos, a estas clases las denominamos conceptos de negocio. Por ejemplo, EN13606 incluye 6 conceptos de negocio: folder, composition, section, entry, cluster y element. Los arquetipos se definen restringiendo alguno de estos conceptos de negocio. De esta forma se sigue el mismo proceso tanto para la definición de nuevos arquetipos a partir del modelo de referencia, como por especialización de alguno ya existente.

La ventana principal de LinkEHR-Ed consta de cuatro componentes:

a. **Vista de árbol del arquetipo**: Aquí se representa la estructura del arquetipo que aparece representada en forma de árbol. Se puede apreciar la estructura, la cabecera, la definición, el idioma y la ontología. El botón  permite cambiar entre la vista técnica y la no técnica.

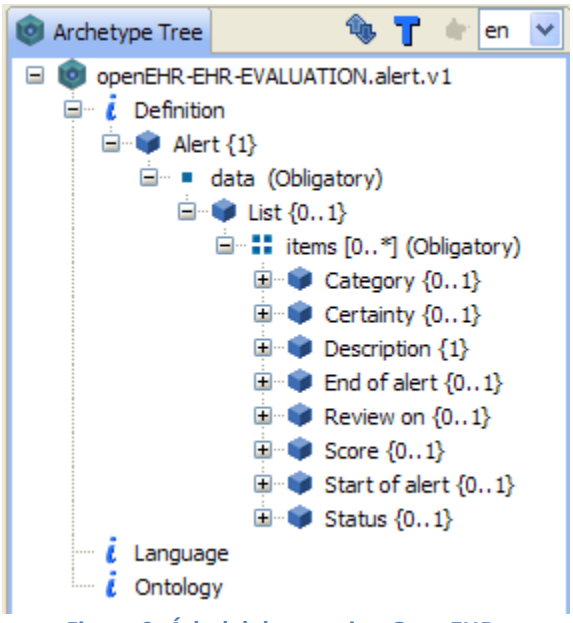


Figura 9: Árbol del arquetipo OpenEHR.

b. **Vista detalle**: En la parte derecha de la pantalla aparecen los distintos formularios para la introducción de datos. Cada tipo de nodo tiene un formulario de datos diferente asociado.

c. **Vista de consola**: esta vista muestra los mensajes de la aplicación. A su vez tiene una pestaña para ver la ruta del nodo seleccionado en el árbol de definición.

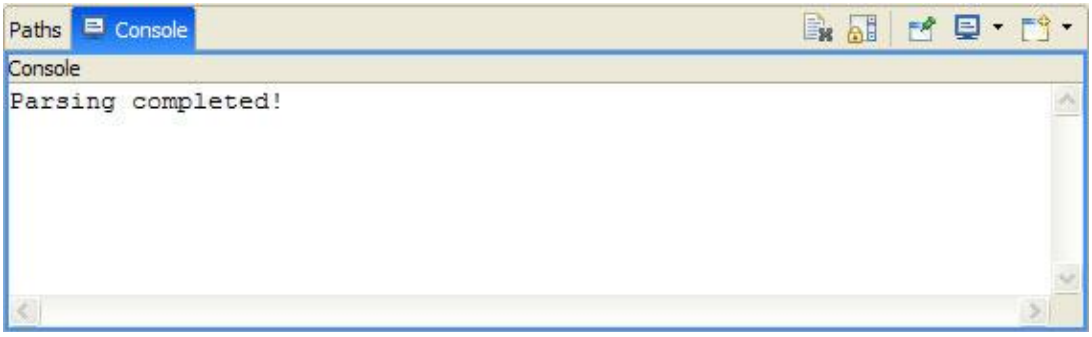


Figura 10: Vista de consola.

d. **Vista ADL**: Cuando el arquetipo ha sido creado, existe la posibilidad de cambiar a la vista ADL para realizar edición directamente sobre el código. Cualquier cambio realizado sobre el código será parseado y revalidado antes de volver a la representación visual. Para cambiar a la vista de ADL se utiliza el botón 

.

openEHR-EHR-OBSERVATION.blood_pressure.v1.adl

```
definition
    OBSERVATION[at0000] matches {
        data matches {
            HISTORY[at0001] occurrences matches {0..1} matches {
                events cardinality matches {1..*; unordered} matches {
                    EVENT[at0006] occurrences matches {0..*} matches {
                        data matches {
                            ITEM_LIST[at0003] occurrences matches {0..1} matches {
                                items cardinality matches {0..*; ordered} matches {
                                    ELEMENT[at0004] occurrences matches {0..1} matches {
                                        value matches {
                                            C_DV_QUANTITY <
                                                property = <[openehr::125]>
                                                list = <
                                                    ["1"] = <
                                                        units = <"mm[Hg]">
                                                        magnitude = <|0.0..1000.0|>
                                                        precision = <|0|>
                                                    >
                                                >
                                            >
                                        }
                                    }
```

Figura 11: Vista ADL del arquetipo de presión sanguínea.

El proceso de edición principal se hace en la rama de definición del árbol del arquetipo. Haciendo clic en cualquier nodo se muestra el formulario con los detalles del nodo. Este formulario dependerá del tipo de nodo. El formulario que se muestra dependerá del tipo de nodo:

- **Atributo**: Hay vistas para editar propiedades de atributos simples o múltiples.

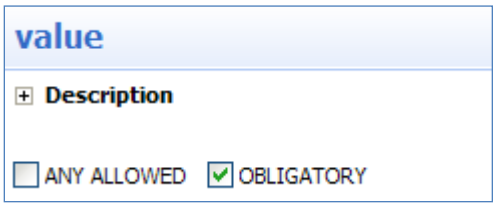


Figura 12: Restricción sobre el valor.

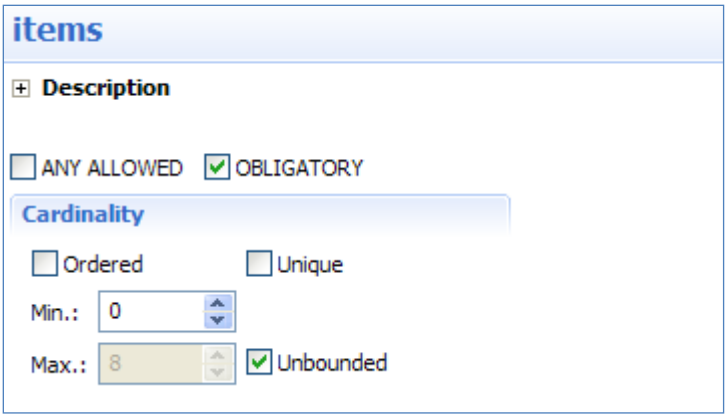


Figura 13: Vista de atributos simples y múltiples.

- **Objeto complejo**: Las ocurrencias de objetos complejos pueden ser modificadas a través del formulario correspondiente. También es posible editar la información de la ontología a través de este formulario.

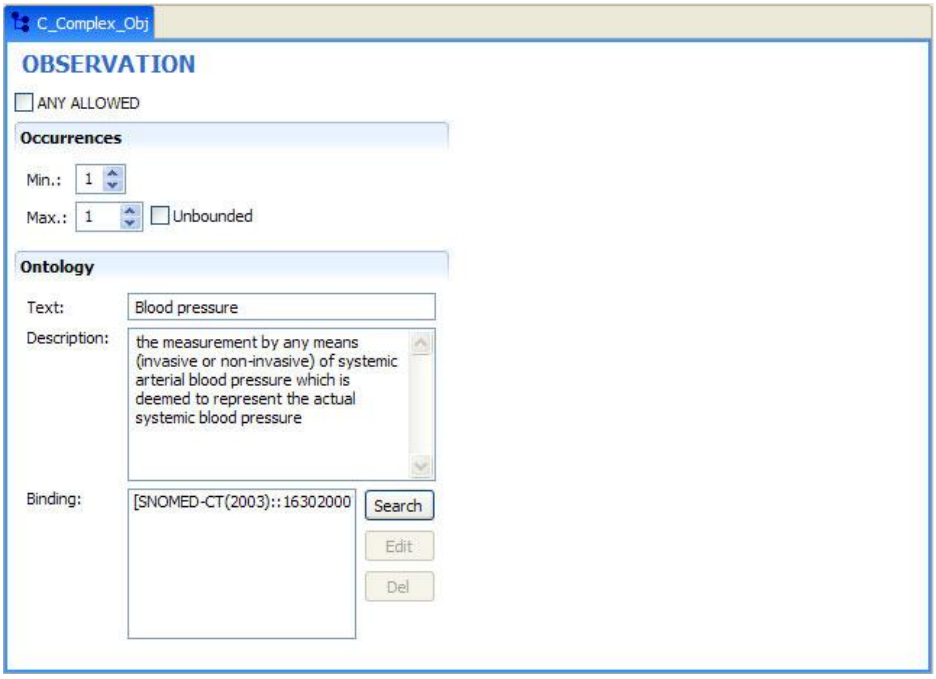


Figura 14: Vista de edición de un CComplexObject.

- **Referencia interna**: Cuando se crea una referencia interna, se proporciona al usuario una lista de nodos destino disponibles con el mismo modelo de referencia. El usuario puede modificar el nodo destino en cualquier momento o ir directamente a su definición.

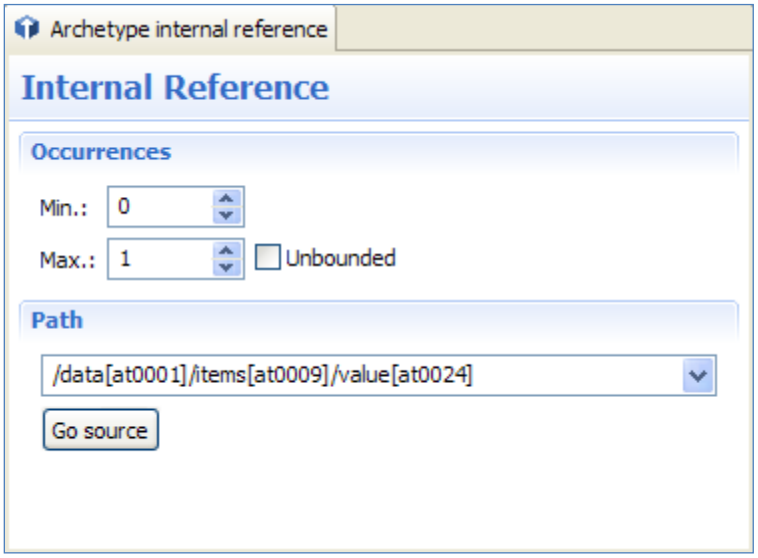


Figura 15: Vista de la referencia interna.

- **Archetype Slot**: En el formulario que se muestra a continuación se puede editar las propiedades del archetype slot.

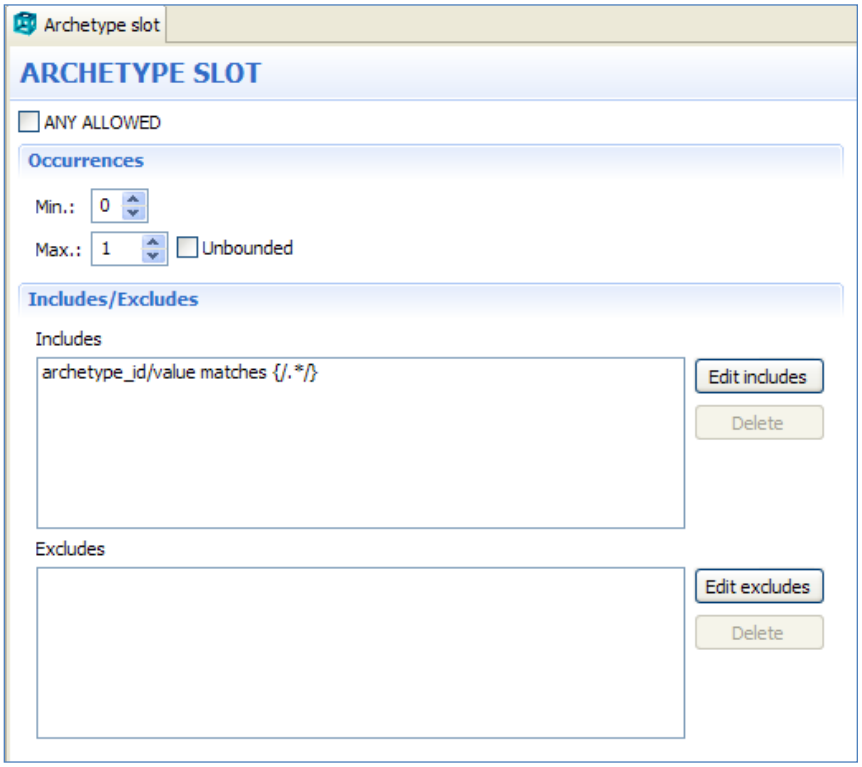


Figura 16: Vista del archetype slot.

- **Primitive Object**: Cada tipo de *primitive object* posee su formulario particular para definir restricciones o valores asumidos.

- **Domain Types**: Al ser LinkEHR-Ed un editor independiente del dominio es difícil diseñar una interfaz para editar tipos específicos de dominio. Debido a ello, no existe interfaz visual para la edición de este tipo de componentes. Aún así, sigue existiendo la posibilidad de editar este tipo de componentes de forma textual en la ventana de ADL.

Para editar el arquetipo se pueden añadir o borrar nuevos objetos y atributos haciendo clic en el botón "Añadir restricción de arquetipo" de la barra de herramientas  o bien haciendo clic en el botón derecho en un nodo sobre el que se quiera llevar a cabo dicha acción. En el menú que se despliega se puede elegir entre crear un objeto, una referencia a un objeto o un archetype slot.

Se puede borrar un nodo seleccionando mediante la opción de borrar del menú que se despliega sobre el nodo de interés.

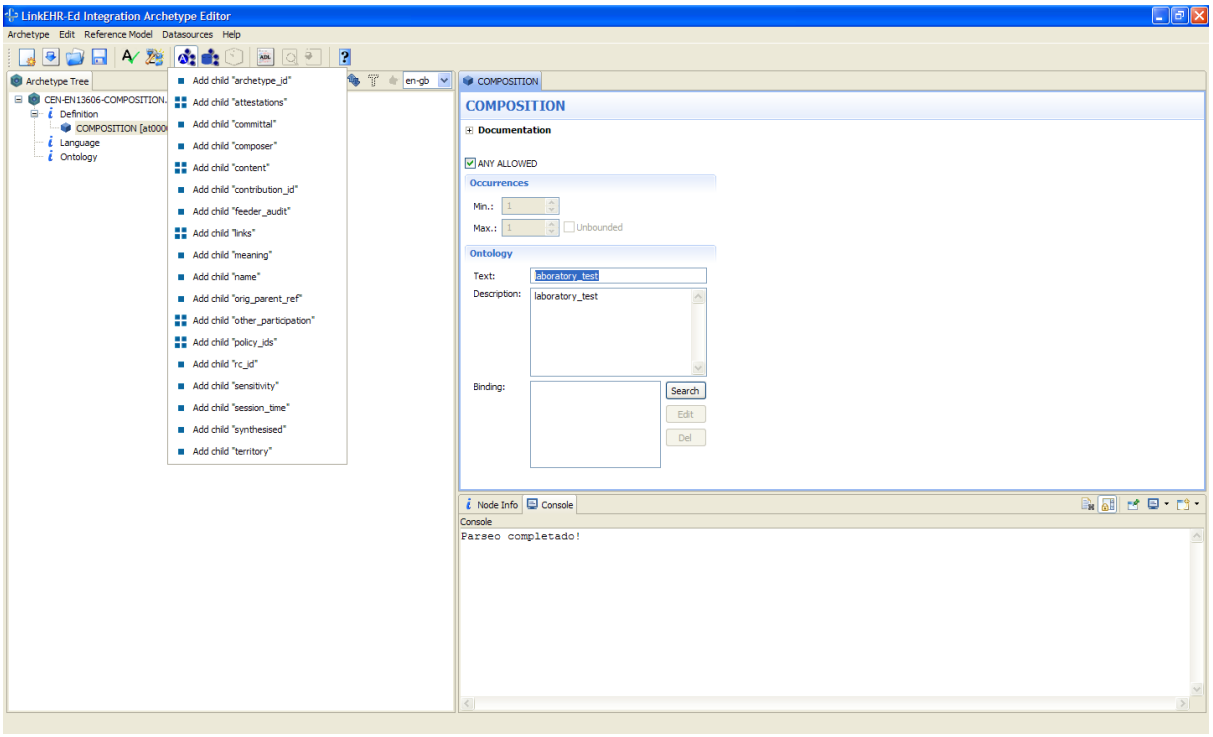


Figura 17: Pulsando sobre el botón "Añadir restricción al arquetipo" se muestra el menú con las acciones disponibles.

3.3. Edición de correspondencias

Como ya hemos comentado en nuestro escenario los arquetipos describen la estructura y contenido de los extractos normalizados de HCE, pero los datos requeridos para generar sus instancias se encuentran almacenados en fuentes de datos no normalizadas. Por tanto, es necesario transformar los datos contenidos en estas fuentes para que sean compatibles con la estructura y etiquetado definido por los arquetipos y el modelo de referencia subyacente. Este problema se conoce en la literatura como traducción o transformación de datos. La transformación de datos requiere a nivel de los esquemas una representación explicita de cómo los esquemas origen y destino se relacionan. En nuestro caso, cómo se relaciona las entidades descritas en el arquetipo (objetos y atributos) con los elementos de datos en las fuentes de datos (por ejemplo, tablas y columnas en el caso de fuentes relacionales o elementos y atributos en el caso de fuentes XML). Estas correspondencias se puede expresar de diversas maneras, por ejemplo por medio de una consulta en un lenguaje de interrogación como SQL o XQuery, un predicado en lógica de primer orden o un conjunto de correspondencias cada una de las cuales relaciona un elemento del esquema origen con un elemento del esquema destino. Por otro lado, a nivel de datos, la transformación de datos trabaja sobre las instancias de los esquemas para transformar instancias de las fuentes en datos en instancias del esquema destino de acuerdo a las correspondencias definidas entre ambas. Estas transformaciones pueden expresarse también de diversas maneras, se puede utilizar un lenguaje específico de transformación como SQL, XQuery o XSLT o un lenguaje de programación de propósito general como Java. En el sector sanitario existen muy pocos trabajos genéricos sobre transformación de datos. A nivel comercial existen algunas herramientas pero con funcionalidad limitada, se centran sobre todo en la generación de HL7 v2.x o mensajes EDI y ninguna de ellas da soporte a los arquetipos. Además los arquetipos no se pueden expresar por medio de DTDs o XML Schemas, por tanto las herramientas de traducción disponibles para estas tecnologías no pueden utilizarse para nuestro propósito.

Los arquetipos imponen una estructura jerárquica en los datos, de hecho podemos modelar los arquetipos como esquemas sobre árboles etiquetados (10). Esto es completamente compatible con XML, formato que utilizamos para representar las instancias. Por esta razón utilizamos XML como modelo canónico para representar el contenido de las fuentes datos, es decir, los datos contenidos en las fuentes se modelan como documentos XML virtuales independientemente de su modelo de datos.

Cuando tenemos un arquetipo en nuestro sistema y deseamos utilizarlo para la extracción y normalización de una parte de la HCE, es necesario establecer una correspondencia entre las fuentes de datos de nuestro sistema de HCE y los nodos del arquetipo. Para llevar a cabo esta operación, LinkEHR-Ed permite establecer las transformaciones necesarias a las fuentes de datos del sistema de HCE y luego asignarlas a un determinado objeto del arquetipo. Veamos a continuación cómo se expresan estas correspondencias.

3.3.1. Tipos de correspondencias

Las correspondencias en LinkEHR-Ed pueden ser de dos tipos: correspondencias de valor y correspondencias de objetos. Las correspondencias de valor definen cómo calcular un valor para un atributo atómico de un arquetipo a partir de un conjunto de valores de la fuente de datos. Están compuestas por un conjunto de pares condición-función. La función es una expresión que calcula un valor a partir de uno o varios valores extraídos de la fuente de datos (se permite el uso de constantes). Mientras que la condición (o filtro) nos indica cuando se puede aplicar la función. Por ejemplo, la correspondencia más simple es aquella que asigna el valor de una columna (en el caso de una fuente relacional) a un atributo atómico del atributo, por ejemplo la correspondencia:

Altura←(true, CCEE.medidas.altura)

Asigna al atributo *Altura* del arquetipo en cuestión el valor contenido en la columna altura de la tabla medidas, perteneciente a la base de datos CCEE (en este caso una base de datos relacional), véase como la condición es *true* y por tanto la función se aplicará siempre. Es posible transformar el valor de la fuente de datos, por ejemplo para pasar de metros a centímetros:

Altura←(true,CCEE.medidas.altura*100)

LinkEHR-Ed permite el uso de un amplio rango de funciones de distintos tipos. Los principales tipos de funciones son:

- **lógicas:** and, or, xor, negación etc.
- **aritméticas:** suma, resta, multiplicación, división etc.
- **De texto:** starts_with, left, right etc.
- **ontológicas:** id, code, archetypeName etc. Sirven para acceder a la sección ontológica de arquetipo y así poder utilizar la información contenida en las instancias del arquetipo.
- **temporales:** current-time, year-from-date, to-ISO-date etc. Incluye funciones para transformar formatos propietario de fechas en fechas conformes con la norma ISO 8601 (Elementos de datos y formatos intercambiables — Intercambio de información — Representación de fechas y horas).

Para terminar, un ejemplo más complejo sería el siguiente, el cual transforma códigos propietarios para representar el sexo de una persona a códigos normalizados:

Filtro	Función
/patient/gender='M' OR /patient/gender='m'	0
/patient/gender='W' OR /patient/gender='w'	1
/patient/gender=0 OR /patient/gender=1	/patient/gender
true	9

Esta correspondencia se puede interpretar como:

If (/patient/gender='M' OR /patient/gender='m') **then** 0
Else if (/patient/gender='W' OR /patient/gender='w') **then** 1
Else if (/patient/gender=0 OR /patient/gender=1) **then** /patient/gender
Else 9

Aunque estas correspondencias son pobres desde el punto de vista semántico presentan algunas ventajas en nuestro escenario. En general los administradores de las fuentes de datos no tienen un conocimiento completo de los esquemas. Por tanto, resulta conveniente que las aserciones utilizadas para describir las correspondencias sean lo más simples posible y que no requieran un conocimiento exhaustivo de los esquemas de las fuentes de datos. En la práctica a los administradores les resulta más sencillo definir qué atributo o expresión se debe utilizar para poblar un atributo del arquetipo que especificar el conjunto de consultas, posiblemente complejas, requeridas para extraer la información necesaria para generar las instancias. Dicho con otras palabras, las correspondencias entre atributos son independientes del diseño lógico de los esquemas de las fuentes de datos. Es indiferente como se hayan agrupados los atributos en tablas (por ejemplo tras un proceso de normalización) o del anidamiento de los elementos en un esquema para XML. Por tanto, no es necesario que la persona encargada de definir las correspondencias tenga que especificar los caminos lógicos de acceso que definen las asociaciones entre los atributos involucrados.

Las correspondencias de valores son fáciles de definir, pero carecen de potencia expresiva para poder describir la semántica de agrupamiento de los datos, es decir, cuando es necesario generar una nueva instancia destino y cómo éstas deben ser agrupadas. Nótese que esto es muy importante ya que estamos trabajando con modelos de datos jerárquicos. LinkEHR-Ed tiene una semántica de agrupamiento por defecto basada en el contexto de los datos, de tal forma que los datos que comparten el mismo contexto (elementos atómicos en los niveles superiores) se agrupan. Es decir, la semántica por defecto está basada en el Forma Normal Particionada (Partition Normal Form o PNF). Los datos de contexto dependen de modelos de referencia y se calculan al vuelo para cada arquetipo.

Si la semántica por defecto no es adecuada para el escenario de transformación, los usuarios pueden definir la suya propia. Es aquí donde entran en juego las correspondencias de objeto (11), las cuales enlazan los objetos complejos del arquetipo con los esquemas origen. Una correspondencia de objeto está formada por un filtro que define una condición sobre el esquema origen y conjunto de caminos, también del esquema origen. Sirven para controlar la creación y agrupamiento, de tal forma que una nueva instancia destino se construye para cada combinación de valores referenciados por los caminos que satisfacen el filtro. Las correspondencias de objeto se pueden enlazar entre sí para formar jerarquías, de tal forma que las correspondencias de nivel superior sirven de contexto para las de nivel inferior, con esto se controla la estructura de los documentos XML resultantes. Por ejemplo, la correspondencia de objeto:

/exploraciones[at0001]/diagnosticos[at0010] ←(/informe/diagnósticos/diagnóstico as $d, $d/código like “10*”)

generará un nuevo objeto de tipo diagnosticos[at0010] para cada valor de /origen/informe/diagnosticos cuyo código de diagnóstico ($d/código) comienza por 10.

Cabe señalar que las correspondencias de valores son obligatorias, es necesario definir una para cada atributo obligatorio en el arquetipo, mientras que las correspondencias de objeto son opcionales, si no están presentes se aplica la semántica de anidamiento por defecto.

El propósito del proyecto fin de carrera descrito en esta memoria es desarrollar una *interface* visual que facilite la edición y gestión de estas correspondencias.

3.4. Generación de XQuery

Cuando el *mapping* de un arquetipo está completo, es decir, todas las entidades obligatorias tienen una correspondencia definida, es posible generar el programa de transformación de datos en XQuery. Estos programas son los que un motor de integración podrá utilizar para normalizar los datos clínicos ya existentes. La plataforma LinkEHR-Ed incluye un motor de integración que es utilizado por el motor de LinkEHR-Ed para generar extractos de HCE normalizados. A grandes rasgos, los pasos a seguir para obtener tales documentos son los siguientes:

1. El extract server recibe una petición para que genere un extracto de información normalizada.

2. LinkEHR Integration Engine proporciona al extract server las vistas unificadas de las fuentes de datos involucradas en la consulta.

3. Con la información obtenida, el extract server aplica la transformación correspondiente al mapping especificado y genera un documento con la información normalizada.

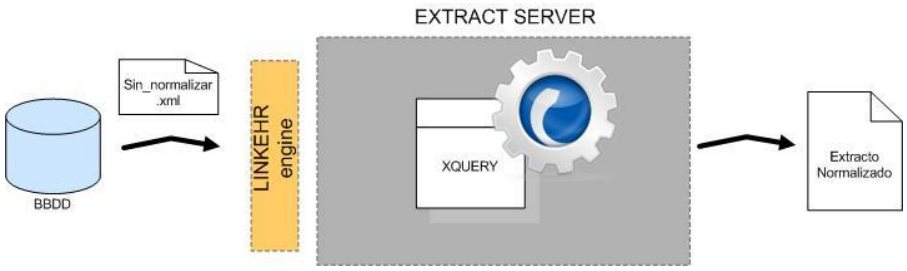


Figura 18: Generación de extracto normalizado.

3.5. LinkEHR Graphical Mapping Manager

Como ya se comentó en apartados anteriores, la principal desventaja de LinkEHR-Ed en la actualidad es su dificultad de uso. El amplio conocimiento de sistemas de información que requiere para poder dar uso a la funcionalidad de correspondencia y edición avanzada hace que el diseño de la herramienta esté más orientado al profesional técnico especialista en TICs de la salud que al clínico.

El principal escoyo es la fase de correspondencia de fuentes de datos del sistema de HCE a LinkEHR-Ed. Requiere que el personal encargado de definir las correspondencias conozca a la perfección tanto el sistema de historia clínica como el arquetipo y las funciones de transformación a utilizar. Es aquí donde este PFC pretende dotar a LinkEHR-Ed de un módulo que permita definir correspondencias mediante un editor gráfico.

GMM (***G**raphical **M**apping **M**anager*) es un módulo que dota al sistema de una representación gráfica de la fuente de datos y las transformaciones que se le aplicarán en el proceso de mapeo. Permite al usuario la construcción de un grafo que representa las fuentes de datos y las transformaciones aplicadas sobre estas. El usuario solamente ha de seleccionar la fuente de datos, constante u operación a incluir en el grafo. El valor de la figura arrastrada (dato o resultado de operación) se une a la siguiente figura aportando así un operando. El último valor se ancla al nodo gráfico que representa al nodo del arquetipo. De esa manera se define la correspondencia entre fuentes de datos y nodos del arquetipo.

3.6. Otras herramientas de edición de arquetipos

Existen diversas herramientas proporcionadas por universidades, fundaciones sin ánimo de lucro y empresas ligadas a ellas. A diferencia de LinkEHR-Ed estas herramientas permiten la definición de arquetipos y plantillas basándose únicamente en un solo modelo de referencia.

Las herramientas de mayor relevancia se exponen a continuación:

Ocean Archetype Editor:

Desarrollado por la empresa Ocean Informatics, proporciona un editor gratuito para arquetipos de OpenEHR. Esta aplicación incluye soporte a metadatos, edición de una forma intuitiva, posibilidad de traducción a cualquier idioma y enlaces a terminologías médicas. Los arquetipos

definidos pueden ser luego utilizados en entornos CEN o traducidos y restringidos para uso en otros formatos clínicos como HL7 CDA.

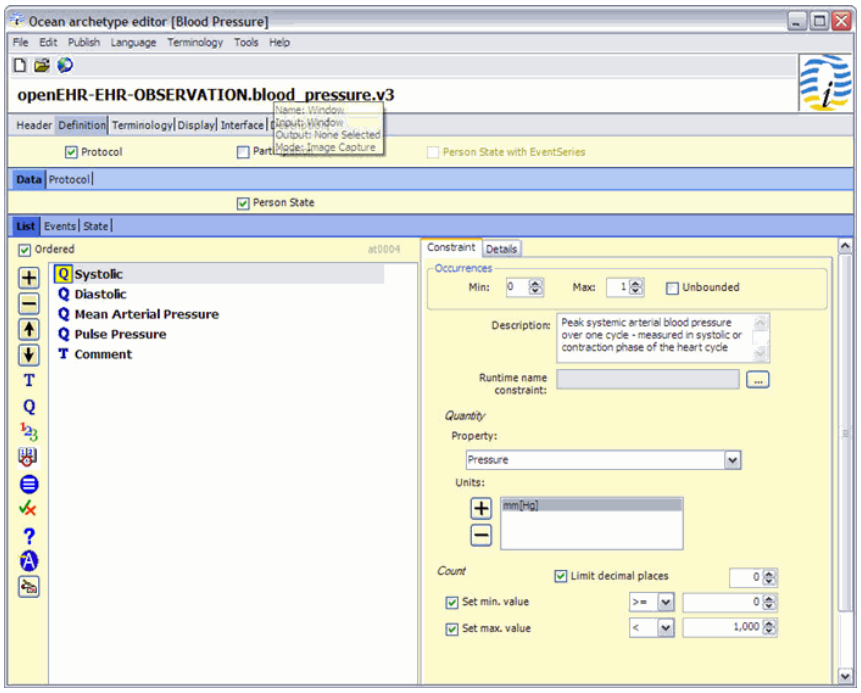


Figura 19: Editor de arquetipos de Ocean.

LIU Archetype editor:

Editor de arquetipos desarrollado en la universidad de Linköping. Esta herramienta fue un proyecto de una tesis de máster, de manera que no es más que un prototipo para uso académico. Cabe decir a su favor que tiene una compatibilidad completa con el editor de OpenEHR siendo los arquetipos con él descritos, plenamente compatibles con cualquier sistema que utilice OpenEHR como modelo de referencia.

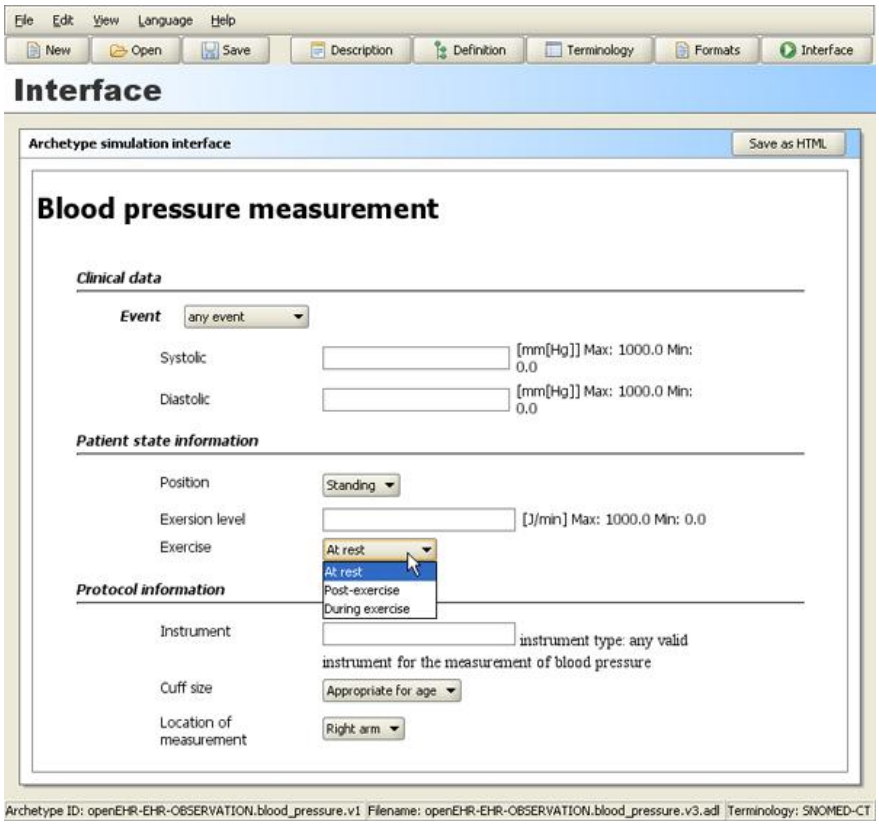


Figura 20: Editor de arquetipos LIU.

OpenEHR Clinical Knowledge Manager:

Aplicación web que actúa como fuente internacional de conocimiento clínico. Mediante tecnología web 2.0 ha conseguido que de forma colaborativa una serie de expertos en el dominio hayan creado un repositorio gratuito y de libre acceso de arquetipos basados en OpenEHR. Estos pueden utilizarse con el objetivo de poder compartir información entre los distintos sistemas de HCE.

Además de las herramientas ya mencionadas, existen algunas otras aplicaciones para la definición de arquetipos y extracción del conocimiento a partir de la información.

Todas estas aplicaciones, junto a LinkEHR-Ed, constituyen el conjunto de tecnologías y métodos que marcan el camino para hacer posible que los sistemas de información clínicos lleguen a ser interoperables.

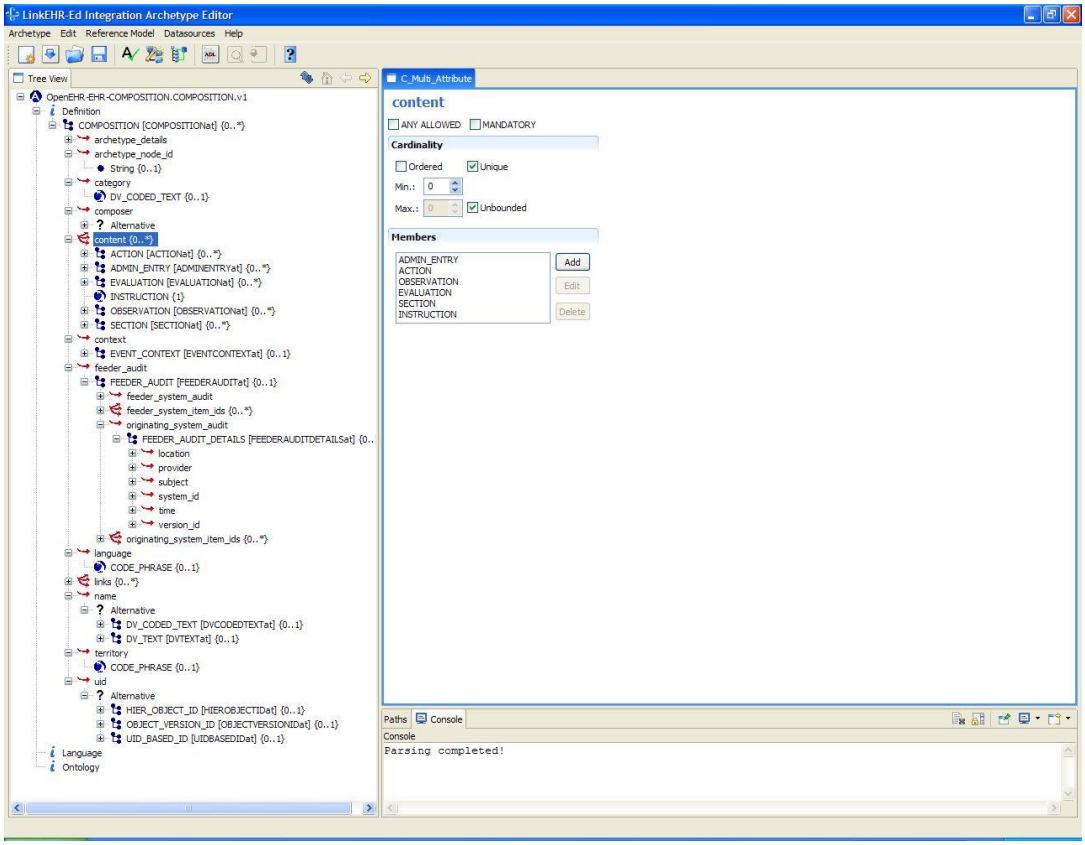


Figura 21: Pantalla de edición de arquetipos de LinkEHR-Ed.

Como comparación de LinkEHR-Ed con OpenEHR, editor de arquetipos líder en la actualidad, cabe decir que este último ha logado una mayor aceptación por parte de facultativos. Esto es debido a su facilidad de uso logrando esconder muchos detalles del modelo de referencia que el clínico no necesita. Esto es posible porque el editor OpenEHR **únicamente** soporta su propio modelo de referencia. Ello facilita enormemente el desarrollo de una *interface* amigable y orientada al facultativo ya que el conjunto de conceptos de negocio es conocido a priori. Como contrapartida, aunque LinkEHR-Ed es un sistema más orientado al técnico, y por ello más complicado de utilizar, es un editor multimodelo. Esto significa que es capaz de importar cualquier modelo de referencia, ya sea CEN, OpenEHR o CDA, y permitir la definición de arquetipos tomando como base el modelo importado. Esta característica dificulta la posibilidad de definir *interfaces* que oculten la complejidad del modelo de referencia, ya que la herramienta está directamente basada en el modelo de arquetipos y por tanto solo es posible definir una *interface* genérica. Para solucionar este problema se propone el uso de *plugins*, uno para cada modelo de referencia. Estos *plugins* contienen metainformación y formularios para una edición amigable de arquetipos basados en un determinado modelo de referencia. En la fecha de redacción de esta memoria estaba disponible el *plugin* para el modelo de referencia de ISO13606. La otra diferencia principal de LinkEHR-Ed con el resto de herramientas para la edición de arquetipos es que es el único que incorpora un módulo para la definición de correspondencias entre los arquetipos y las fuentes de datos. Este módulo permite definir correspondencias de alto nivel declarativas entre las fuentes de datos de un sistema particular y un arquetipo. Estas correspondencias son compiladas en programas de transformación de datos expresados en XQuery que transforman los datos ya existentes en documentos XML compatibles con el modelo de referencia y el arquetipo en cuestión.

4. Requerimientos

A continuación se describirán los principales requisitos de **GMM**.

4.1. Requisitos generales

La principal necesidad de LinkEHR-Ed era el desarrollo de un módulo que facilitara la definición de correspondencias entre el esquema de la fuente de datos (esquema origen) y las entidades del arquetipo (esquema destino). Se hacía pues necesario el diseño de un editor que, de forma intuitiva, permitiera al usuario la especificación de tales correspondencias de forma sencilla y visual. GMM es un editor **WYSIWYG** (***W****hat* ***Y****ou* ***S****ee* ***I****s* ***W****hat* ***Y****ou* ***G****et*), esto quiere decir que el diagrama que construimos se corresponde totalmente con la cadena que se obtendrá y validará como correspondencia final a guardar.

El editor debe permitir al usuario arrastrar componentes situados en una paleta de herramientas anclada al lateral del lienzo. Tras arrastrar los componentes principales que conforman el grafo, se debe permitir al usuario la edición y modificación de los componentes creados. Después, el usuario podrá conectar los distintos componentes de forma que se alimente a las operaciones y dirijan los resultados de estas a donde interese para, finalmente, llevar el último resultado al nodo del arquetipo al que interese asociar la correspondencia.

4.2. Descripción de los Casos de Uso

En este apartado se describen de una manera más específica los diferentes casos de uso y funcionalidades que el editor podrá llevar a cabo.

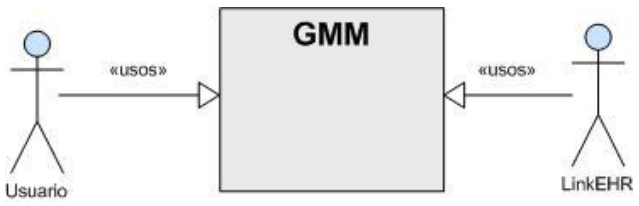


Figura 22: Delimitación del sistema según actores.

Las 3 acciones más generales que se pueden llevar a cabo en el mapeo son las mostradas en la figura: abrir el editor, editar correspondencias y cerrar el editor.

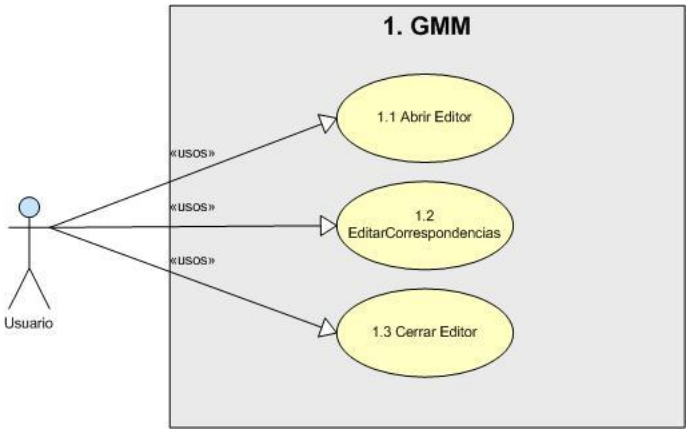


Figura 23: Caso de uso 1.

CU 1.1 Abrir Editor

El lanzamiento del editor supone acceder al *Archetype Constraint* seleccionado por el usuario y consultar si tenía algún trabajo en curso para recuperarlo o bien, si no lo tenía, mostrar la primera de las correspondencias en su lista.

Caso de Uso	Abrir Editor
Actores	Usuario
Tipo	Básico
Propósito	Lanzar el editor para visualización o edición de mapeos.
Resumen	El usuario lanza el editor para visualizar o definir los mapeos de algún Archetype Constraint.
Precondiciones	Debe existir un Archetype Constraint definido.
Flujo Principal	

	Entradas del Actor	Respuesta del Sistema
1	Seleccionar el nodo del que verá sus correspondencias.	
2	Seleccionar acción de abrir el editor.	
3		Recuperar el nodo seleccionado.
3		Consultar correspondencias para ese nodo.
4		Mostrar esquema de la primera correspondencia de la lista.

Subflujos	
Excepciones	E1- *No existe un Archetype Constraint seleccionado:* Informar al usuario de la invalidez de la selección.

CU 1.2. Editar Correspondencias

La edición de correspondencias engloba a tres actividades bien diferenciadas. Éstas, como se muestra en la figura, son la construcción del grafo para la correspondencia, la validación de la correspondencia y el salvado de la correspondencia.

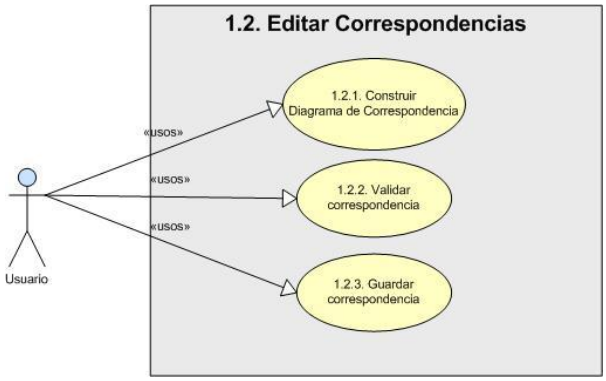


Figura 24: Caso de uso Editar Mapeos.

CU 1.2.1. Construir Diagrama de Correspondencia

La construcción del diagrama de correspondencia se lleva a cabo permitiendo al usuario las operaciones de inserción de constante, operación y fuente de datos en el diagrama, así como la interconexión de los componentes incluidos.

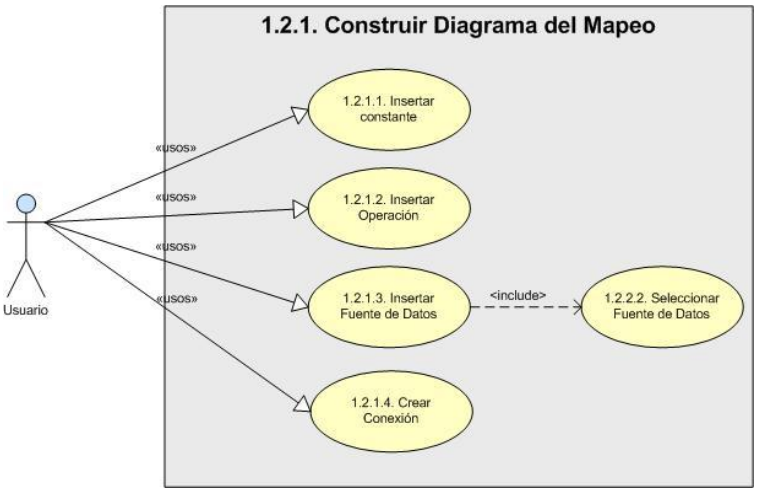


Figura 25: Caso de uso 1.2.1.

CU 1.2.1.1. Insertar Constante

Supone la inclusión de un objeto genérico en el grafo. Este objeto permitirá editar sobre él un valor cualquiera que puede ir desde una constante numérica a una cadena de texto o ruta de datos.

<table>
<tr><td>Caso de Uso</td><td>Insertar constante.</td></tr>
<tr><td>Actores</td><td>Usuario.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Insertar una constante en el grafo de la correspondencia.</td></tr>
<tr><td>Resumen</td><td>El usuario selecciona la opción constante y la arrastra al editor.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Seleccionar constante en la paleta y llevarla al lienzo.</td><td></td></tr>
<tr><td>2</td><td></td><td>Crear el comando de construir constante.</td></tr>
<tr><td>3</td><td></td><td>Ejecutar comando de construcción.</td></tr>
<tr><td>3</td><td>Editar valor de la constante.</td><td></td></tr>
<tr><td>4</td><td></td><td>Guardar valor de la constante.</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td></td></tr>
</table>

CU 1.2.1.2. Insertar Operación

Permite al usuario la inserción de una figura correspondiente a una operación o función que se llevará a cabo sobre los operandos que se conecten a ella.

<table>
<tr><td>Caso de Uso</td><td>Insertar operación.</td></tr>
<tr><td>Actores</td><td>Usuario.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Insertar una operación en el grafo de la correspondencia .</td></tr>
<tr><td>Resumen</td><td>El usuario selecciona la opción operación y la arrastra al editor.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Seleccionar operación y llevarla al lienzo</td><td></td></tr>
<tr><td>2</td><td></td><td>Crear el comando de construir operación</td></tr>
<tr><td>3</td><td></td><td>Consultar operación en fichero de funciones</td></tr>
<tr><td>3</td><td></td><td>Crear operación</td></tr>
<tr><td>4</td><td></td><td>Ejecutar comando de creación</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td></td></tr>
</table>

CU 1.2.1.3. Insertar Fuente de Datos

Mediante esta función el usuario será capaz de seleccionar una ruta sobre una fuente de datos que haya sido previamente seleccionada y asociada al editor mediante la función del caso de uso 1.2.2.2.

<table>
<tr><td>Caso de Uso</td><td>Insertar Fuente de Datos.</td></tr>
<tr><td>Actores</td><td>Usuario.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Insertar una fuente de datos en el grafo de la correspondencia.</td></tr>
<tr><td>Resumen</td><td>El usuario selecciona la opción operación y la arrastra al editor.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.
Se debe haber seleccionado una fuente de datos con la que trabajar.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Selecciona la opción fuente de datos en la paleta y llevarla al lienzo.</td><td></td></tr>
<tr><td>2</td><td></td><td>Crear comando para la creación de la fuente de datos</td></tr>
<tr><td>3</td><td></td><td>Abrir el diálogo de selección de la fuente de datos.</td></tr>
<tr><td>4</td><td>Seleccionar la entidad que se desea de la fuente de datos.</td><td></td></tr>
<tr><td>5</td><td>Aceptar.</td><td></td></tr>
<tr><td>6</td><td></td><td>Ejecutar comando de creación</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td>S1 El usuario selecciona una ruta de la fuente de datos previamente seleccionada que será el valor que tome la figura fuente de datos en el grafo. Esta fuente ha de aceptarse como válida.</td></tr>
<tr><td>Excepciones</td><td>E1 No existe ninguna fuente de datos asociada al editor para seleccionar la ruta. Se mostrará un mensaje al usuario.</td></tr>
</table>

CU 1.2.1.4. Crear conexión

El uso de esta funcionalidad es la conexión de los diversos componentes del grafo de manera que conduzcamos los valores adecuados a las operaciones que deseemos. De la misma manera será posible alimentar a los nodos objetivo de la correspondencia con el resultado de una operación o bien alimentar a una operación con el resultado de otra.

<table>
<tr><td>Caso de Uso</td><td>Crear Conexión.</td></tr>
<tr><td>Actores</td><td>Usuario.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Crear una conexión entre dos nodos del grafo del mapeo.</td></tr>
<tr><td>Resumen</td><td>El usuario une dos nodos que sean válidos para ser conectados entre sí.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.
El nodo fuente y destino deben haberse creado previamente y ser semánticamente correctos para permitir la unión.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Selecciona la de conexión en la paleta y llevarla al lienzo</td><td></td></tr>
<tr><td>2</td><td></td><td>Crear comando para la creación de la conexión.</td></tr>
<tr><td>3</td><td>Unir la fuente con el destino deseados.</td><td></td></tr>
<tr><td>4</td><td></td><td>Ejecutar comando de creación</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td>E1 El destino de la conexión no es válido: Se le indica al usuario que el destino de su conexión no es válido y no se le permite completar la conexión de los dos nodos.</td></tr>
</table>

CU 1.2.2. Validar Correspondencias

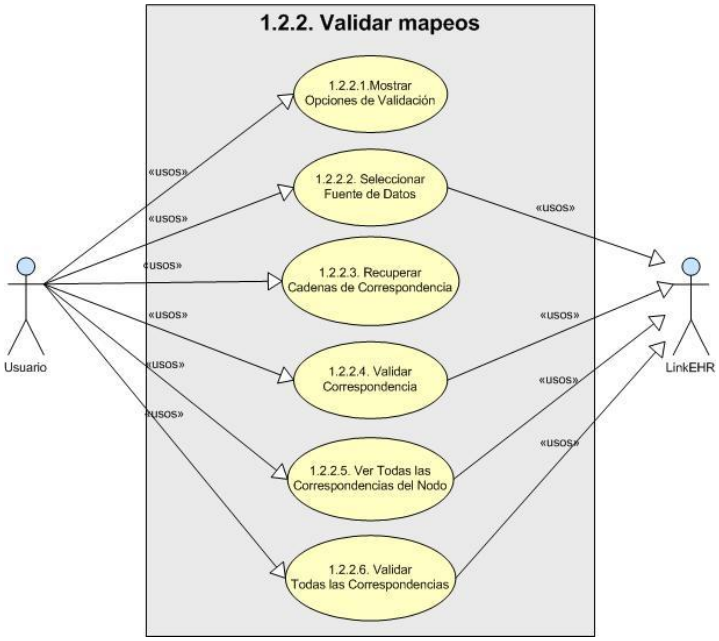


Figura 26: Caso de uso 1.2.2.

CU 1.2.2.1. Mostrar Opciones de Validación

Permitirá al usuario la visualización de las funciones disponibles sobre las cadenas recuperadas para convertirlas en correspondencias.

<table>
<tr><td>Caso de Uso</td><td>Mostrar opciones de validación</td></tr>
<tr><td>Actores</td><td>Usuario</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Poner a disposición del usuario las opciones de validación, recuperación y consulta de correspondencias.</td></tr>
<tr><td>Resumen</td><td>El usuario visualizará las opciones para la correcta validación, recuperación y consulta de correspondencias.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Inicialización del editor o cambio de instancia del editor.</td><td></td></tr>
<tr><td>2</td><td></td><td>Mostrar la vista de consulta y validación de correspondencias conforme a la instancia del editor que se está visualizando.</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td></td></tr>
</table>

CU 1.2.2.2. Seleccionar Fuentes de Datos

El usuario podrá asociar al editor una de las fuentes de datos que hayan sido importadas en LinkEHR-Ed. A partir de esta fuente de datos se seleccionará la entidad de la fuente de datos cuya ruta aparecerá en el diagrama mediante la función del caso de uso 1.2.1.3.

Caso de Uso	Seleccionar fuente de datos.
Actores	Usuario, LinkEHR kernel.
Tipo	Inclusión.
Propósito	Seleccionar la fuente de datos asociada al editor a partir de la que se seleccionarán las rutas de correspondencia.
Resumen	El usuario selecciona una de las fuentes de datos que la aplicación mantiene como importadas.
Precondiciones	El editor debe haber sido inicializado correctamente. Las fuentes de datos disponibles para seleccionar deben haber sido importadas a LinkEHR correctamente.
Flujo Principal	
Subflujos	
Excepciones	

	Entradas del Actor	Respuesta del Sistema
1	Activar opción de selección de fuente de datos.	
2		Lanzar diálogo de selección.
3	Seleccionar fuente de datos.	
4		Guardar selección.

CU 1.2.2.3. Recuperar Cadenas de Correspondencia

Permite recuperar cada una de las cadenas de correspondencia concerniente a cada nodo mediante una exploración del grafo construido a partir de la conexión de constantes, operaciones y fuentes de datos.

<table>
<tr><td>Caso de Uso</td><td>Recuperar cadenas de correspondencia.</td></tr>
<tr><td>Actores</td><td>Usuario</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Recuperar a partir del grafo construido las cadenas de texto que podrán ser salvadas como correspondencia.</td></tr>
<tr><td>Resumen</td><td>Se recuperan y muestran al usuario cada una de las cadenas que se corresponden con los grafos anclados a los diferentes nodos de la representación del arquetipo.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.

Se debe haber construido un grafo cuyo final sea uno de los nodos de la figura que representa a la sección del arquetipo seleccionada.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Activar opción de recuperación de cadenas.</td><td></td></tr>
<tr><td>2</td><td></td><td>Recorrer para cada nodo en orden down-top el grafo del mapeo construyendo la cadena a la que este corresponde.</td></tr>
<tr><td>3</td><td></td><td>Completar la tabla con las correspondencias recuperadas tanto para la condición como para la correspondencia.</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td></td></tr>
</table>

CU 1.2.2.4. Validar una Correspondencia

El usuario será capaz de validar si las cadenas de texto recuperadas tras explorar el grafo son válidas. Para ello se comprobará si el par filtro/función es válido interrogando al *kernel*.

<table>
<tr><td>Caso de Uso</td><td>Validar una correspondencia.</td></tr>
<tr><td>Actores</td><td>Usuario, LinkEHR kernel.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Validar una correspondencia recuperada del grafo.</td></tr>
<tr><td>Resumen</td><td>Consultar al LinkEHR kernel si una determinada correspondencia recuperada del editor es válida tanto en filtro como en función.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.

Haber recuperado correctamente un mapeo del editor.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Activar opción de validación de la correspondencia de interés</td><td></td></tr>
<tr><td>2</td><td></td><td>Validar contra el kernel de la aplicación.</td></tr>
<tr><td>3</td><td></td><td>Informar sobre la corrección de la correspondencia.</td></tr>
<tr><td>4</td><td></td><td>Ofrecer salvar la correspondencia.</td></tr>
<tr><td>5</td><td>Aceptar salvar la correspondencia</td><td></td></tr>
<tr><td>6</td><td></td><td>Salvar correspondencia</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td>S1 Si la correspondencia validada es correcta se ofrecerá la opción de salvarla.</td></tr>
<tr><td>Excepciones</td><td>E1 Correspondencia incorrecta: Se informará al usuario de la incorrección de la correspondencia y su motivo.</td></tr>
</table>

CU 1.2.2.6. Validar todas las correspondencias de un nodo

El usuario podrá validar todas las correspondencias recuperadas del diagrama. Visualizará cuales de las correspondencias son correctas o no.

<table>
<tr><td>Caso de Uso</td><td>Validar todas las correspondencias.</td></tr>
<tr><td>Actores</td><td>Usuario, LinkEHR kernel.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Validar todas las correspondencias existentes en el diagrama.</td></tr>
<tr><td>Resumen</td><td>Consultar al LinkEHR kernel si todas y cada una de las correspondencias recuperadas del editor son válidas tanto en filtro como en función.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.

Haber recuperado correctamente todas las correspondencias del editor.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Activar opción de validación múltiple.</td><td></td></tr>
<tr><td>2</td><td></td><td>Validar contra el kernel de la aplicación todas y cada una de las correspondencias previamente cargadas en la tabla.</td></tr>
<tr><td>3</td><td></td><td>Informar sobre la corrección de todas las correspondencias.</td></tr>
<tr><td>4</td><td></td><td>Ofrecer salvar las correspondencias.</td></tr>
<tr><td>5</td><td>Aceptar salvar las correspondencias.</td><td></td></tr>
<tr><td>6</td><td></td><td>Salvar correspondencias.</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td>E1 Alguna Correspondencia Incorrecta: Se informará al usuario de de la existencia de alguna correspondencia incorrecta.</td></tr>
</table>

CU 1.2.2.7. Visualizar Todos los Correspondencias de un Nodo

El usuario visualizará de forma comprensible cada una de las correspondencias existentes para un nodo.

<table>
<tr><td>Caso de Uso</td><td>Ver todas las correspondencias de un nodo.</td></tr>
<tr><td>Actores</td><td>Usuario, LinkEHR kernel.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Mostrar al usuario en una vista fácilmente comprensible todas las correspondencias asociadas a un determinado Archetype Constraint.</td></tr>
<tr><td>Resumen</td><td>Se recuperarán y mostrarán los todas las correspondencias existentes para un Archetype Constraint.</td></tr>
<tr><td>Precondiciones</td><td>El editor debe haber sido inicializado correctamente.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Activar opción de ver todas las correspondencias referentes a un nodo</td><td></td></tr>
<tr><td>2</td><td></td><td>Recuperar cada una de las correspondencias que un nodo pueda tener.</td></tr>
<tr><td>3</td><td></td><td>Mostrar en la vista de validación todas las correspondencias referentes al nodo seleccionado.</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td></td></tr>
</table>

CU 1.3. Cerrar Editor

Al cerrar el editor es posible guardar el estado en el que queda el diagrama para ofrecer al usuario la posibilidad de continuar su trabajo en un futuro.

<table>
<tr><td>Caso de Uso</td><td>Cerrar editor.</td></tr>
<tr><td>Actores</td><td>Usuario.</td></tr>
<tr><td>Tipo</td><td>Básico.</td></tr>
<tr><td>Propósito</td><td>Cerrar el editor gráfico de correspondencias.</td></tr>
<tr><td>Resumen</td><td>El usuario cierra el editor después de acabar su trabajo con él.</td></tr>
<tr><td>Precondiciones</td><td>El editor debió abrirse sin percances.</td></tr>
<tr><td>Flujo Principal</td><td>
<table>
<tr><th></th><th>Entradas del Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>Cerrar editor.</td><td></td></tr>
<tr><td>2</td><td></td><td>Preguntar al usuario si desea salvar la edición en curso.</td></tr>
<tr><td>3</td><td>Confirmar salvado del editor.</td><td></td></tr>
<tr><td>3</td><td></td><td>Persistir la edición en curso</td></tr>
<tr><td>4</td><td></td><td>Cerrar</td></tr>
</table>
</td></tr>
<tr><td>Subflujos</td><td></td></tr>
<tr><td>Excepciones</td><td>E1 Problemas en el salvado del trabajo actual: Error al persistir el editor con el que se estaba trabajando. El usuario pierde el esquema que no hubiese salvado como correspondencia.</td></tr>
</table>

5. Métodos

5.1. Tecnologías

La elección de las tecnologías utilizadas para la realización del proyecto se ha basado en la búsqueda de una herramienta donde prime la facilidad de uso, la facilidad de aprendizaje y el uso intuitivo de la herramienta de edición de correspondencias con una formación mínima. Se concluyó que para la consecución de esas metas la mejor opción era el desarrollo de un editor que permitiera al usuario la construcción de un grafo que incluiría las operaciones y funciones a realizar sobre las fuentes de datos. A su vez, este grafo acabaría en un nodo sumidero que no es más que el nodo del arquetipo al que van a parar los datos, una vez han pasado por las transformaciones necesarias. Con lo anterior se conseguiría una representación visual de la correspondencia que se hace sobre las fuentes de datos y la condición aplicada a este.

Framework

El proyecto ha sido llevado a cabo mediante la utilización del entorno de desarrollo *Eclipse*, un IDE extensible en funcionalidades mediante *plugins* (12). El *framework* de Eclipse es actualmente el más versátil del mercado. Permite el desarrollo de aplicaciones, no solo en JAVA, lenguaje para el que fue creado originalmente, si no para lenguajes tan diferentes como C, C++, COBOL o PHP. Es el usuario quien decide, aprovechando la arquitectura basada en *plugins* de eclipse, crear el entorno a su medida instalando todos los complementos que pueda necesitar. Como principal desventaja frente a otros entornos de desarrollo encontramos las carencias a la hora de buscar documentación y ayuda sobre el uso de la plataforma. Si bien esta existe en forma de artículos y foros, se encuentra muy diseminada haciendo que el desarrollador pierda gran parte del tiempo recopilando los artículos que necesita consultar.

GEF:

El ***G****raphical* ***E****diting* ***F****ramework* (13) (14) (15) de Eclipse es un marco compuesto de varias librerías para la construcción de editores que permiten la modificación del modelo objetual subyacente.

GEF está compuesto por la librería gráfica *draw2d* y una organización en diferentes paquetes que permiten preservar la independencia entre el modelo, la representación gráfica de este y las *Parts* que son los objetos intermedios que relacionan a ambos. Con una arquitectura ***M****odel* ***V****iew* ***C****ontroller* permite separar claramente la representación de la capa de negocio y los procesos de control de la aplicación. Su principal desventaja es que

el *framework* presenta una de las curvas de aprendizaje con mayor pendiente que existen. Esto hace que no sea posible aprenderlo gradualmente como sucede con la mayoría de tecnologías, si no que, incluso para el desarrollo de un editor sencillo, sea necesario un conocimiento bastante completo del *framework*.

Draw2D:

Es una librería gráfica que permite la representación de diversas formas con diferentes *layouts* y conexiones entre ellas. Aporta distintas formas de renderizado, *scrolling* y *thumbnail*. Hasta ahora la principal dificultad era casarlo con la capa de negocio de la aplicación debido a la dificultad de mantener una estructura ordenada y coherente pero esto ha quedado resuelto por GEF.

5.2. Arquitectura

5.2.1. El Patrón MVC

La arquitectura seguida, y que ha venido marcada por la tecnología utilizada ha sido ***M**odel **V**iew **C**ontroller* (16). El patrón de diseño software MVC nació en 1979. Fue creado por Trygve Reenskaug para la construcción de las *interfaces* de usuario del Smalltalk-80. Hasta ese momento las aplicaciones que se desarrollaban eran principalmente monolíticas. Esto quiere decir que interfaz, lógica y datos se englobaban en una sola capa causando grandes problemas para el mantenimiento y evolución de los sistemas. El patrón de diseño MVC intenta evitar esto separando las distintas partes del sistema.

El patrón se distribuye en tres partes principales:

Modelo:

Es la representación del dominio de interés. Contiene la lógica de la aplicación que evoluciona entre diferentes estados al recibir un estímulo del controlador. Al producirse un cambio en el estado del modelo, este informará a la vista para que actualice la interfaz de forma que muestre información coherente con el nuevo estado. Además el modelo es el que encapsula la capa de persistencia, de forma que es aquí donde se realiza el acceso a datos y el arranque a partir de datos almacenados. MVC no especifica nada más sobre como relacionar la persistencia con la lógica de la aplicación.

Vista:

Es la representación del modelo. Usualmente se corresponde con la GUI (***G**raphical **U**ser **I**nterface*). Como MVC desacopla el modelo lógico de la representación de este, varias vistas pueden coexistir para un mismo modelo. Cada una de las vistas existentes tiene capacidad propia de actualizarse a sí misma.

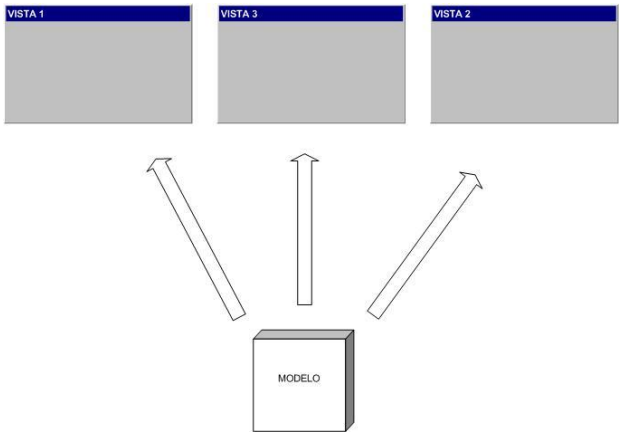


Figura 27: Diferentes vistas para un mismo modelo.

Controlador:

Es el encargado de recibir las entradas al sistema y actuar en consecuencia, realizando la llamada correspondiente al modelo. Al igual que ocurre con las vistas, es fácil cambiar un controlador por otro modificando así la forma de responder del sistema sin necesidad de realizar cambios ni en el modelo ni en la vista. Normalmente los controladores se anidan de forma jerárquica especializando el comportamiento a la acción a la que estos responden.

En la figura que se muestra a continuación se muestra el ciclo que sigue una interacción del usuario con la aplicación cuando ésta se basa en el patrón MVC.

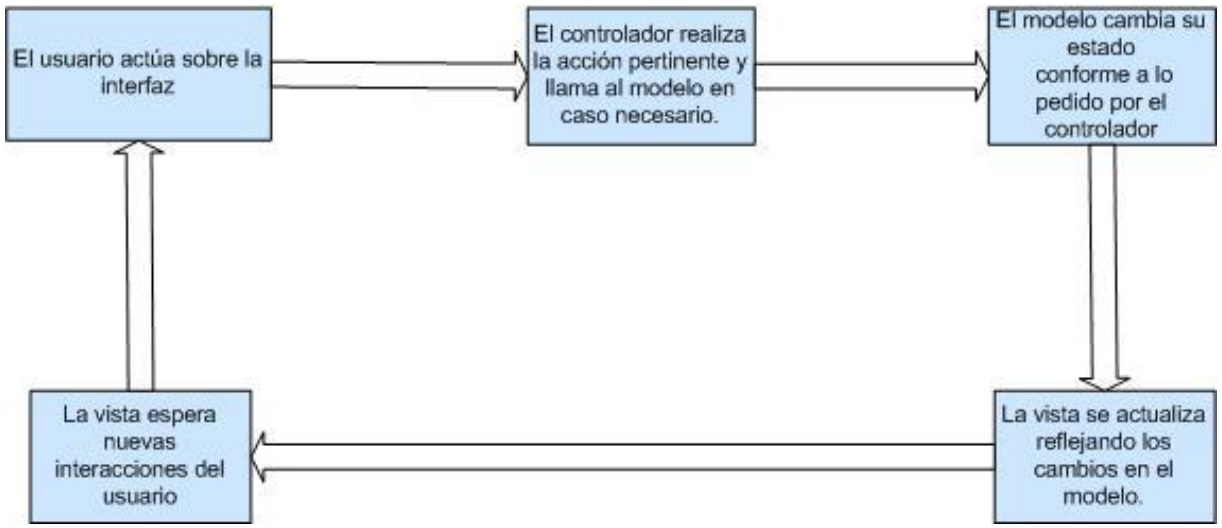


Figura 28: Flujo de acciones MVC.

5.2.2. Aplicación de MVC a GMM

El patrón MVC ha sido aplicado a GMM de la forma y estructura que marca GEF.

Modelo:

Cada objeto representado en el modelo de GMM se corresponde con un objeto útil para la correspondencia en el modelo de referencia de LinkEHR-Ed. Por ello podemos encontrar operaciones de transformación para las correspondencias, fuentes de datos, constantes, objetos complejos etc.

Cada objeto del modelo mantiene las conexiones y relaciones que tiene con otros objetos del diagrama. De esa forma, un objeto operación mantiene una lista con las conexiones que le llegan para ser capaz de identificar a sus operandos y la conexión que sale de él como resultado.

Finalmente, el modelo es el que contiene las funciones lógicas necesarias para el correcto comportamiento del editor. Un ejemplo son los métodos de ordenación situados en la operación, los cuales se encargan de la correcta recuperación de los operandos dependiendo de en que parte de la figura estén anclados.

Vista:

La vista se corresponde principalmente con el paquete *figures*. Cada uno de los objetos de este paquete es capaz de generar una figura que representa a uno de los objetos del modelo.

Los objetos de este paquete se encargan del diseño de la figura correspondiente asignándole el *layout*, decoraciones y anidación de las distintas formas que definen la forma del objeto a representar. A su vez se encargan de definir los anclajes a los que será posible unir una conexión. De esa forma cuando se intenta crear una conexión, la figura analiza si en las proximidades de la posición del ratón hay algún punto donde se pueda anclar una conexión para permitir o no esa acción.

Controlador:

Los controladores se corresponden en GEF con las *EditParts*. Generalmente existe un *EditPart* para cada uno de los objetos a representar. Como se muestra en el diagrama las *EditPart* son los objetos que hacen de nexo entre modelo y vista. Son también los objetos responsables de la edición y, en muchos casos, especializan su comportamiento mediante la delegación a los llamados *EditPolicies*, objetos encargados de definir y controlar la mayoría de las tareas de edición.

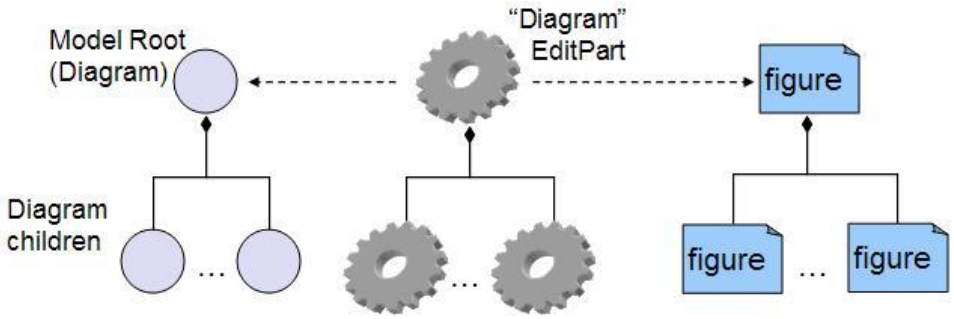


Figura 29: Relación del modelo con las vistas mediante los EditPart (help.eclipse.org).

Por último queda hacer mención a los comandos de edición. Estos objetos son invocados cuando el usuario hace uso de alguna de las funcionalidades del editor. Contienen la lógica para realizar un cambio que afectará al modelo y que, a su vez, deberá reflejarse en la vista. Al realizar las acciones a través de comandos podemos almacenar cada uno de los comandos llevados a cabo en una pila de forma que si en algún momento nos conviene deshacer alguna de las acciones no hay más que ver el último comando almacenado y ejecutar su método deshacer.

La figura siguiente muestra a grandes rasgos la interacción entre el usuario y el editor a través de *Requests* (peticiones) al controlador para que este genere el comando. La ejecución de este será la que lleve a cabo alguna de las tareas de la edición.

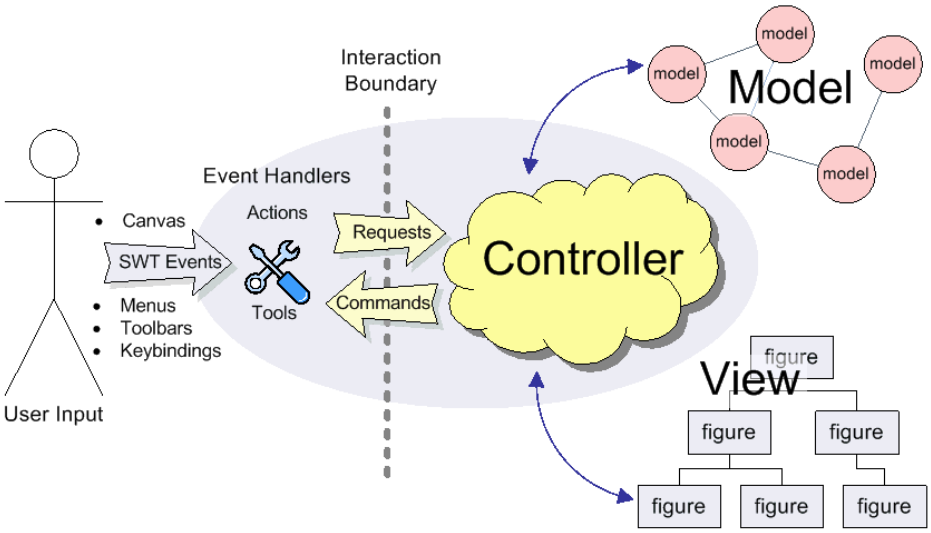


Figura 30: MVC aplicado a GEF.

6. Diseño

El diseño de aplicación se ha acometido siguiendo el modelo que marcan los ejemplos que Eclipse proporciona como muestra de GEF.

6.1 Paquetes que componen la aplicación

GMM es un editor compuesto por diez paquetes:

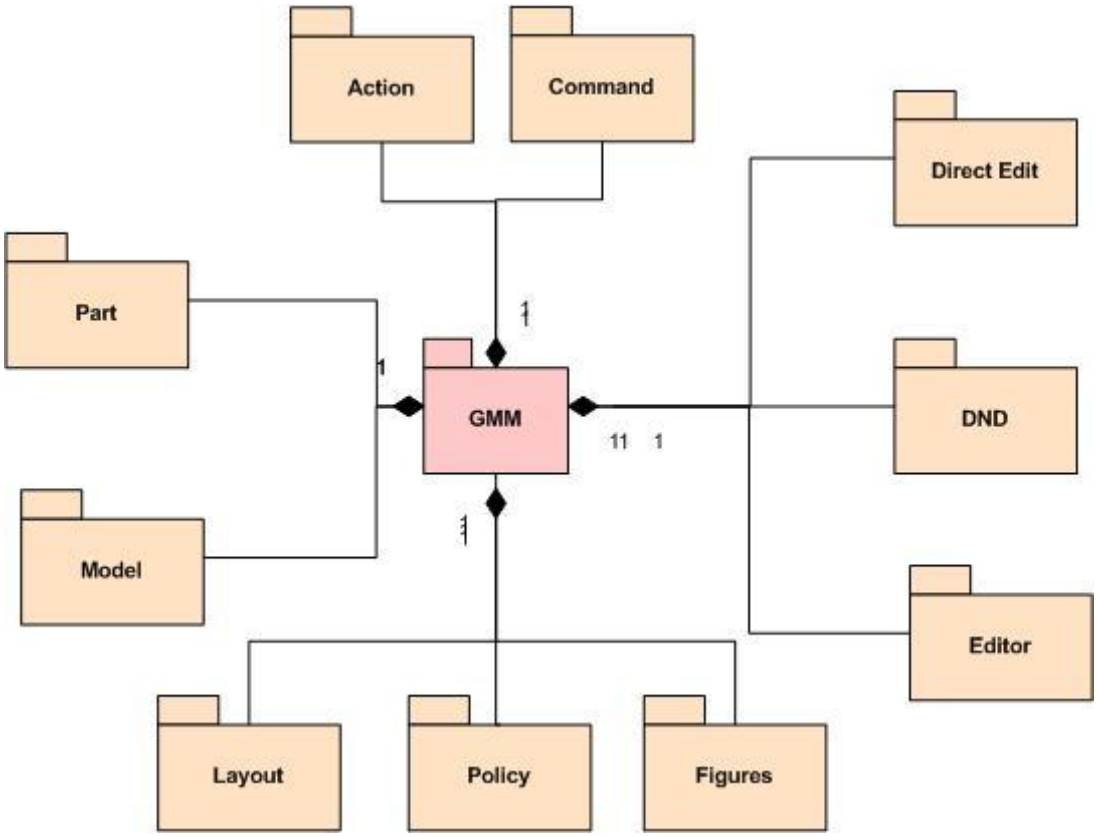


Figura 31: Paquetes que componen a GMM.

Paquete Action:

En este paquete están contenidas las acciones generales que el usuario puede llevar a cabo sobre el editor. Las principales son el cambio del modo de **layout** manual/automático y mostrar/ocultar los diálogos de ayuda a la edición de correspondencias.

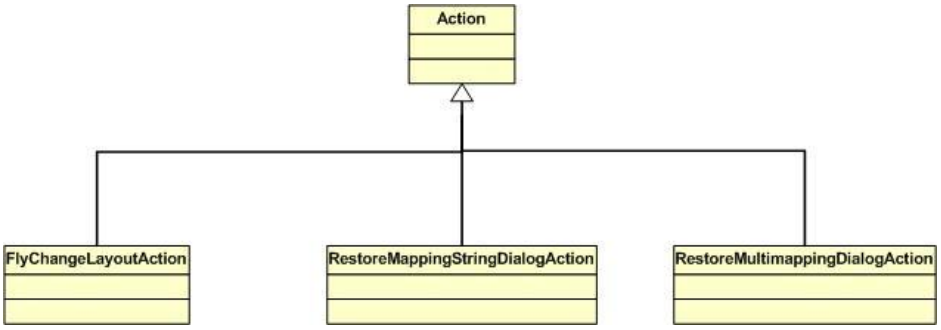


Figura 32: Paquete Action.

SchemaContextMenuProvider: Esta acción es la encargada de gestionar las opciones que se muestran en menú contextual que aparece al pulsar botón derecho sobre el editor. Extiende la clase *ContextMenuProvider* de GEF, de manera que permite añadir un menú contextual de forma estructurada y bien integrada en la aplicación

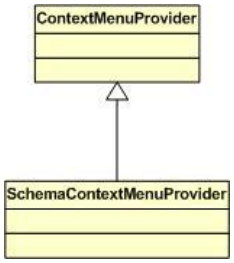


Figura 33: SchemaContextMenuProvider.

SchemaActionBarContributor*:* Esta acción se encarga de realizar una contribución a la barra de menú de LinkEHR-Ed. Mediante ella se examina el nodo del árbol que aparece seleccionado y se muestra una figura con la parte del arquetipo que cuelga de él. Las acciones con las que se ha

contribuido a la barra de LinkEHR-Ed son el botón para activar/desactivar el *layout* automático y el combo box que contiene los niveles de zoom.

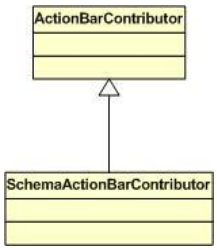


Figura 34: SchemaActionBarContributor.

Command:

El paquete *Command* es el contenedor de las clases que son invocadas cada vez que se realiza una acción de edición. Cada clase representa una acción que puede ser consultada, para comprobar si es posible ejecutarla, llevar a cabo la ejecución de ésta o deshacer la acción. Las clases de este paquete contienen métodos que se corresponden con las tres acciones que se les puede solicitar. Los comandos se usan para encapsular y combinar los cambios en el modelo de la aplicación.

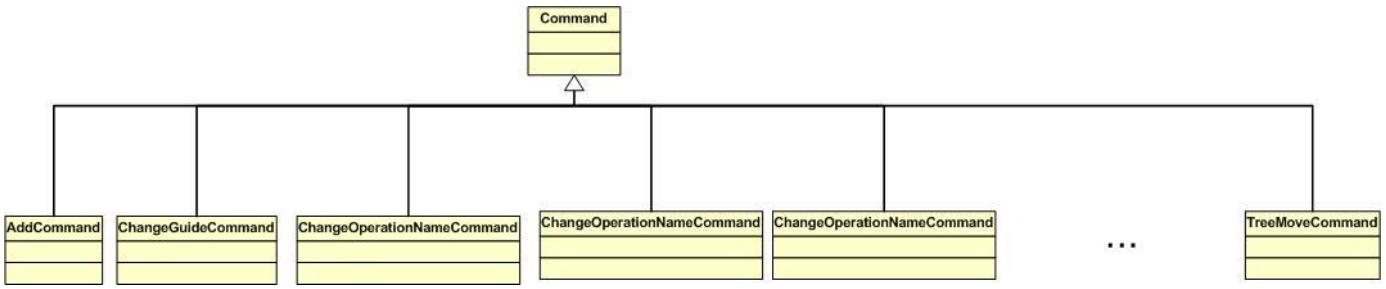


Figura 35: Paquete Command.

Direct Edit:

Este paquete engloba los gestores, validadores y localizadores de las áreas que permiten la edición directa. Es aquí donde, por ejemplo, si se desea cambiarle el nombre o valor a una constante se definen los valores aceptados, el área donde se editará y la posición de ésta.

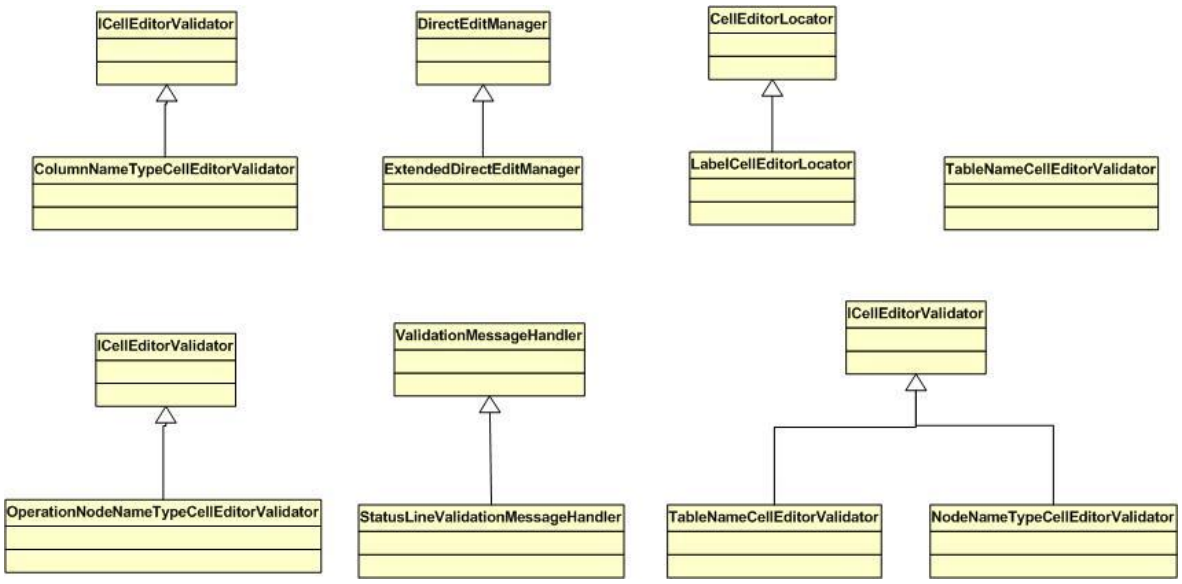


Figura 36: Paquete DirectEdit.

DND:

Las dos clases existentes son:

DataEditDropTargetListener: Esta clase es la encargada de proveer de un *listener* para arrastrar nuevos elementos al panel de dibujo del editor.

DataElementCreationFactory: Provee de una *Factory* para la creación de nuevos objetos a partir de la paleta.

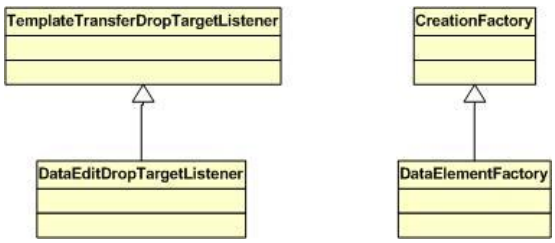


Figura 37: Factorías de creación de datos.

Editor:

Este paquete contiene un conjunto general de clases necesarias para el correcto funcionamiento del editor. Las de mayor importancia se enumeran a continuación.

ContentCreator: Es la encargada de crear el contenido inicial del editor. Para ello consulta que nodo es el seleccionado para mostrar la correspondencia de cada uno de sus hijos.

SchemaPaletteViewerProvider: Inicializa la acción de *Drag & Drop*.

DatasourceSelector: Permite seleccionar la fuente de datos a partir de la cual seleccionaremos la ruta de extracción de la información.

LayoutSwitchingClass: Es la clase encargada de gestionar los cambios en el tipo de *layout*.

OperationsProvider: Lector del archivo XML que contiene las operaciones y funciones de correspondencia.

MappingTreeFigureCreator: Crea la figura que representa el fragmento de arquetipo que cuelga del nodo seleccionado.

SchemaDiagramEditor: Implementación del editor mediante extensión de la clase *GraphicalEditorWithFlyoutPalette*.

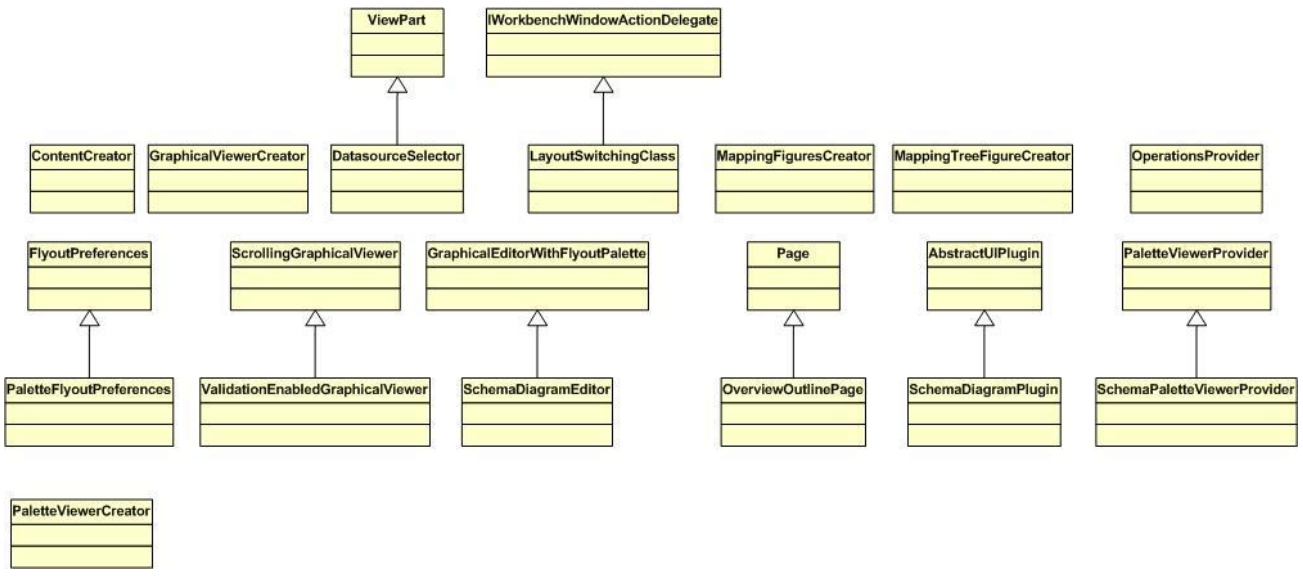


Figura 38: Contenidos del paquete editor.

Figures:

En este paquete están contenidas las implementaciones de las figuras que se dibujan en el editor. Además aquí aparecen también los anclajes que les son asignados a las figuras para que sea posible iniciar o terminar una conexión.

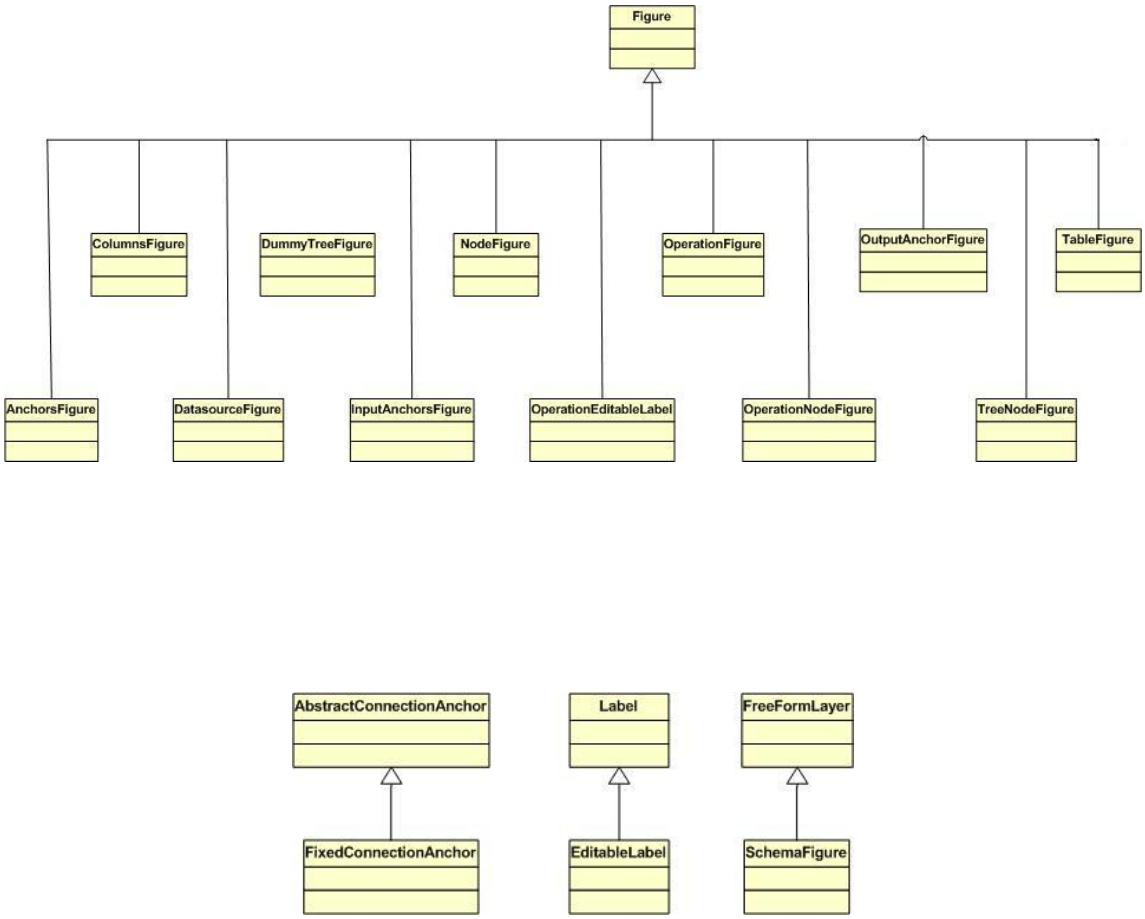
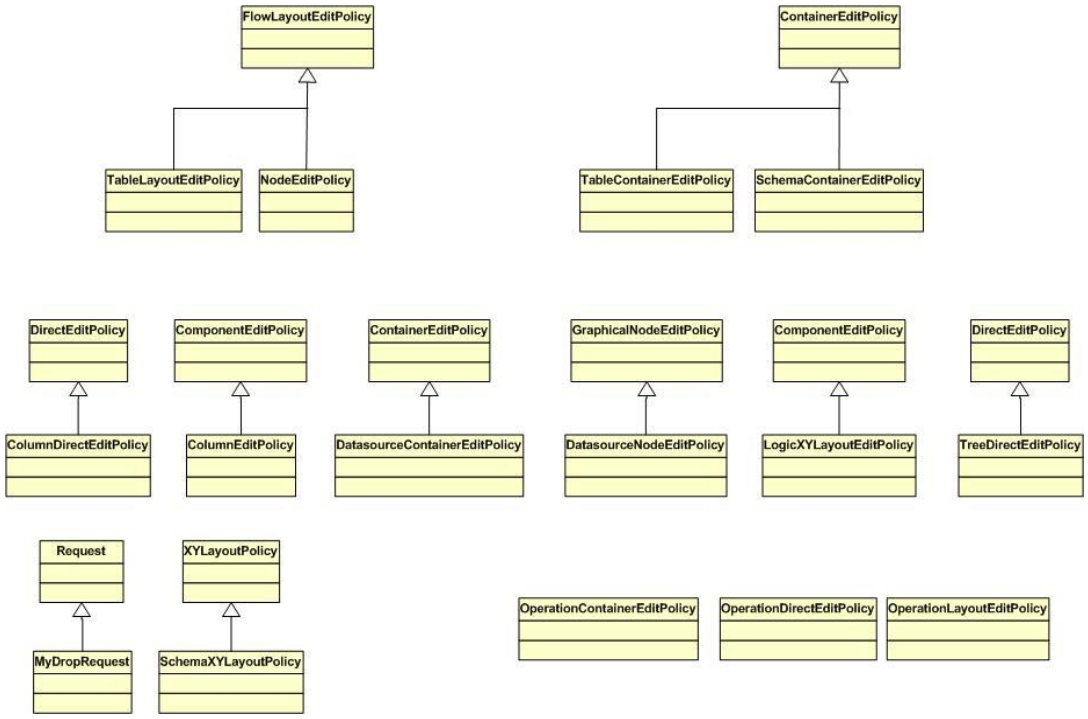


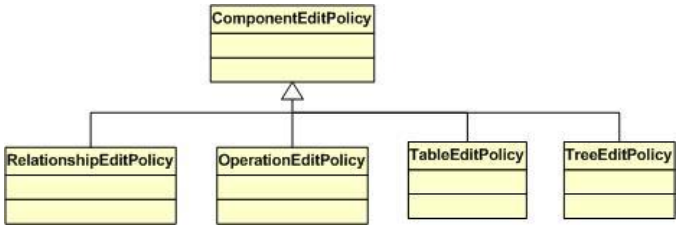
Figura 39: Paquete figure.

Policy:

Este paquete contiene las clases encargadas de modelar las políticas, es decir los elementos que describen la manera en que se comporta el editor.

Existen diversas especializaciones en la descripción de las políticas. Se describen aspectos tan diversos como el *layout* que cada una de las figuras utilizará, bajo qué condiciones se permitirá anclar conexiones o qué tipo de valores admitirá una celda.





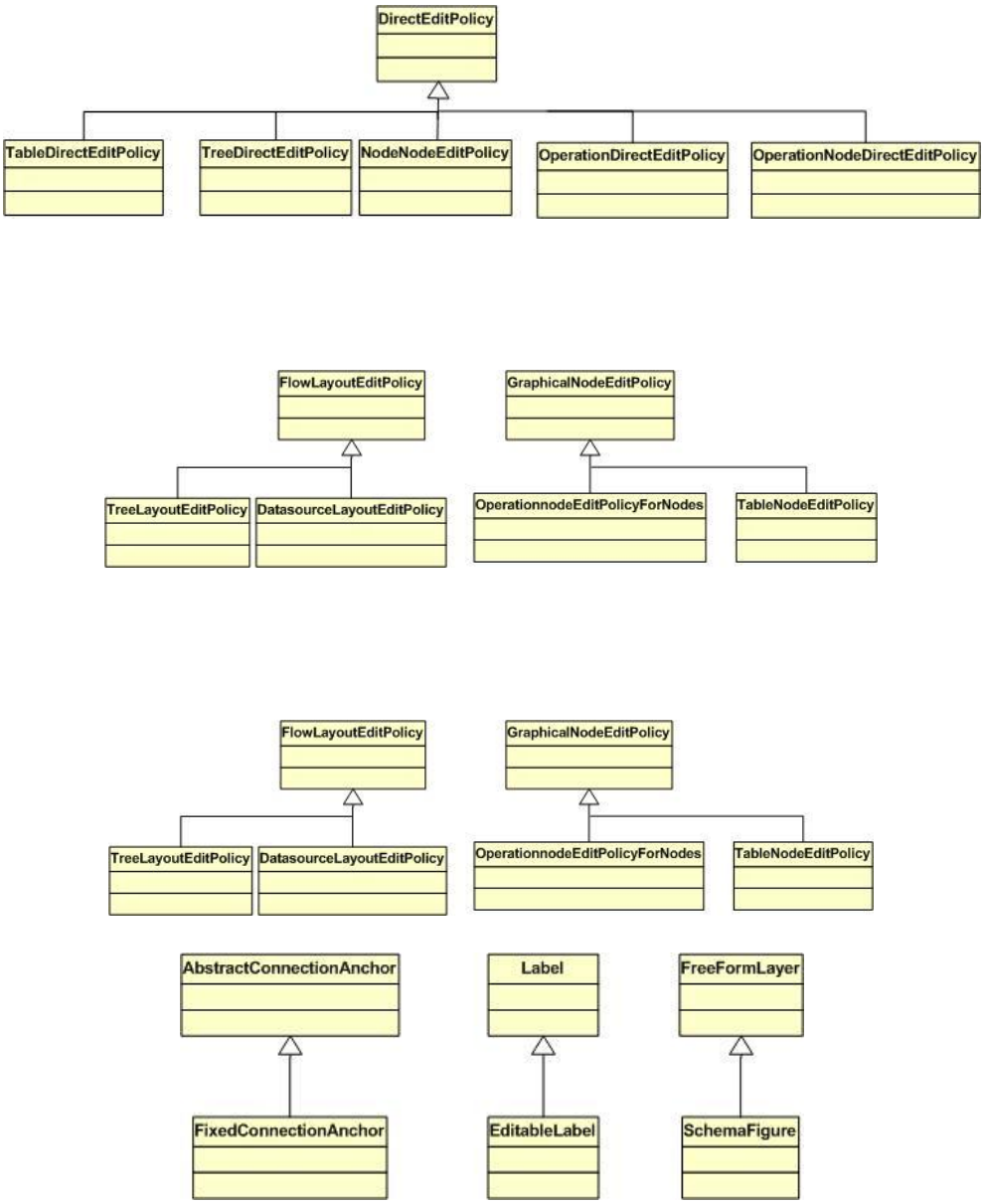


Figura 40: Paquete Policy.

Layout:

Cada una de las clases que definen layout de figuras y componentes.

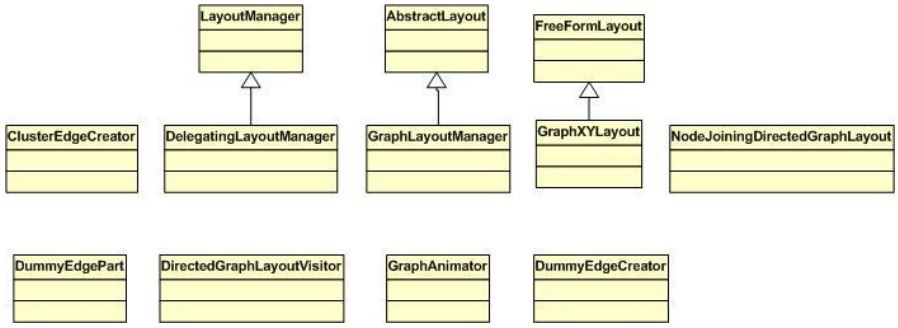
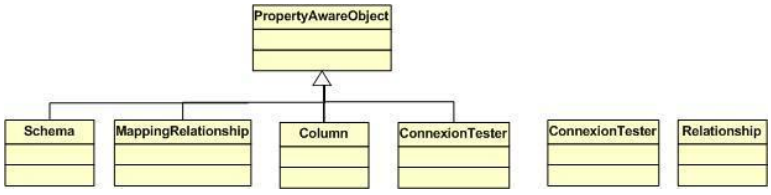


Figura 41: Paquete Layout.

Model:

Este paquete define cada uno de los objetos del modelo de la aplicación. Cada uno de estos objetos se corresponde con uno de los objetos de LinkEHR-Ed que forman parte de una correspondencia. Los principales son *Tree* (define al arquetipo), *Table* (representa a una constante) y *Datasource* (una fuente de datos relacional o XML) etc.



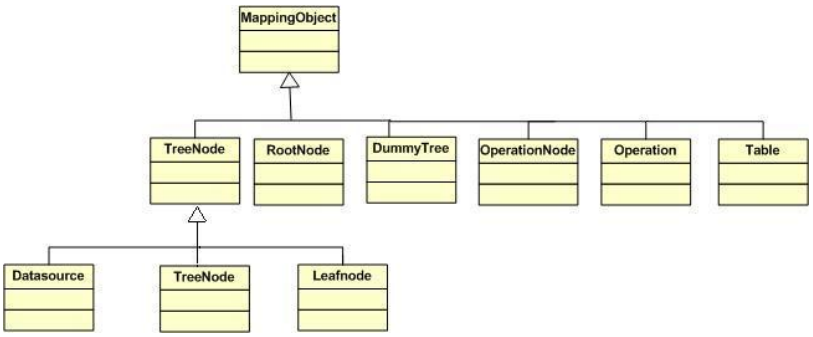
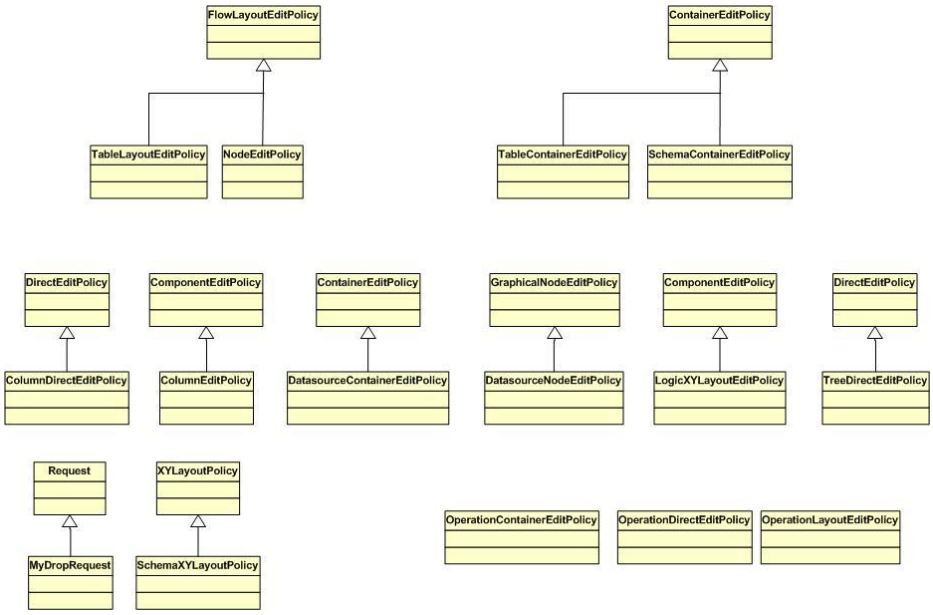


Figura 42: Paquete model.

Part:

Quedan contenidos en este paquete los objetos que hacen de puente entre el modelo y la vista. Las *Part* son las encargadas de mantener una referencia tanto a la vista como al modelo asociado a esa vista. A su vez son los objetos que contienen una relación de las conexiones entrantes y salientes al objeto asociado. Es en cada una de las *Part* donde se instalan las políticas que afectarán al objeto. Estos objetos pueden además consultar que hijos tiene a nivel de modelo el objeto al que corresponde el *Part*.



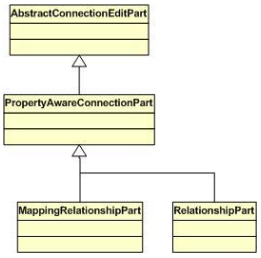
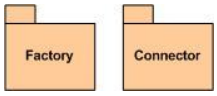


Figura 43: Paquete Part.



El paquete ***Factory*** contiene la clase *SchemaEditPartFactory*, la cual es una factoría que se encarga de crear la *Part* correspondiente a partir de un modelo del objeto.

El paquete ***Connection*** contiene los objetos *Achor*. Estos son los anclajes que situados en una figura permiten tener puntos de conexión. A su vez se encargan de detectar si el usuario está en sus proximidades para hacer una recomendación sobre el lugar donde anclar la conexión.

6.2 Edit Parts

Cada parte editable del modelo necesita ser asociada con una instancia de *EditPart*. Normalmente existe una correspondencia uno a uno entre las clases de la jerarquía del modelo y las clases de la jerarquía de las *EditPart*. En los casos en los que un *EditPart* es a su vez un *GraphicalEditPart*, significa que, además de tener asociado un componente del Modelo, tiene asociado un componente gráfico. Como en GEF, modelo y vista están completamente desacoplados, toda la coordinación entre ambos ha de hacerse a través de un *EditPart*.

Para aislar la fuente de interacción GEF usa los mecanismos *Request*. Los mecanismos de invocación de funciones de la GUI serán los encargados de crear una *Request* en consonancia a la acción activada. A su vez estos mecanismos ejecutarán la acción realizando las llamadas pertinentes a las *EditPart* que corresponda. El proceso paso a paso es el descrito a continuación:

1. Decidir que *EditPart* es la que está involucrada examinando el *Part* de la figura situada debajo del ratón.
2. Mostrar interacción mientras se realiza la acción. Un ejemplo de ello es el área sombreada al arrastrar un componente del editor.
3. El *EditPart* crea el *Command* correspondiente, el cual, una vez ejecutado, será el que realice un cambio en el modelo. Si se intenta realizar una interacción no permitida o para la que no existe un *Command* asociado, la interfaz mostrará el símbolo de no permitido.
4. Por último existe un API que indica al *EditPart* que haga algo. Esto no es algo que tenga que repercutir en el modelo siempre. Por el contrario, puede ser una ejecución sin repercusiones como abrir un determinado diálogo.

6.3 Edit Policies

Puesto que las *EditParts* no realizan la edición directamente han de servirse de las *EditPolicies* (Políticas de Edición). Cada política se centra en una sola tarea de edición o como mucho varias que estén estrechamente relacionadas. De esta forma se consigue reutilizar una política para la implementación de varias *EditParts*. Cuando uno de los métodos de edición de la *EditPart* es invocado, la *EditPart* delega en su política para que lleve a cabo la acción.

6.4 Ciclo de vida de una EditPart

1. **Creación**: La mayoría de las *EditPart* se crean en la *factory* de la vista. Esta normalmente es invocada por la *EditPart* del objeto padre. Es en esta fase donde se le asigna un objeto del modelo a la *EditPart*.

2. **Añadiendo al Diagrama**: En esta fase se le asigna a la *EditPart* un padre (setParent(...)), una figura (*setFigure()*) y una notificación (*addNotify()*). Inmediatamente después se activa mediante la invocación del método *activate()*. De esta forma la *EditPart* se vuelve activa y comienza a 'escuchar' eventos del editor que recaigan sobre ella. Además activará las políticas, hijos y conexiones dependientes de ella.

3. **Uso**: La *EditPart* permitirá su uso conforme a lo especificado en sus políticas y comandos.

4. **Destrucción**: Cuando su edición termine, la vista invocará el método *deactivate()* de forma que la *EditPart* quedará inutilizada y a merced del colector de basura de Java.

6.5 La Paleta de Herramientas

Las herramientas disponibles en la paleta manejan la mayoría de eventos del editor. Un objeto *EditDomain* es el encargado de controlar la herramienta que está seleccionada.

Las herramientas están implementadas como máquinas de estados. Los eventos de SWT son los que provocan la transición entre estados. Las acciones que las herramientas pueden llevar a cabo son: la obtención de comandos de las *EditParts*, ejecución de comandos de la pila, actualizar el cursor del ratón etc.

Las herramientas se activan colocándolas en el *EditDomain* y solo una de ellas puede estar activa en un instante determinado. Al seleccionar una herramienta de la paleta lo que se está haciendo es indicarle al Editor cual es la herramienta activa para que actúe en consecuencia.

Un buen ejemplo de herramienta lo encontramos en la herramienta de selección. Esta herramienta suele ser la herramienta por defecto en la mayoría de los editores. La mayor parte de su vida la gasta delegando acciones en diversos manejadores. Un ejemplo es el *DragTracker* que es el manejador activado al arrastrar alguna de las componentes del editor, modificar su tamaño o implementar una conexión. En la figura se muestra un diagrama extraído del tutorial de GEF que proporciona eclipse en el que se especifica el funcionamiento del *DragTracker*.

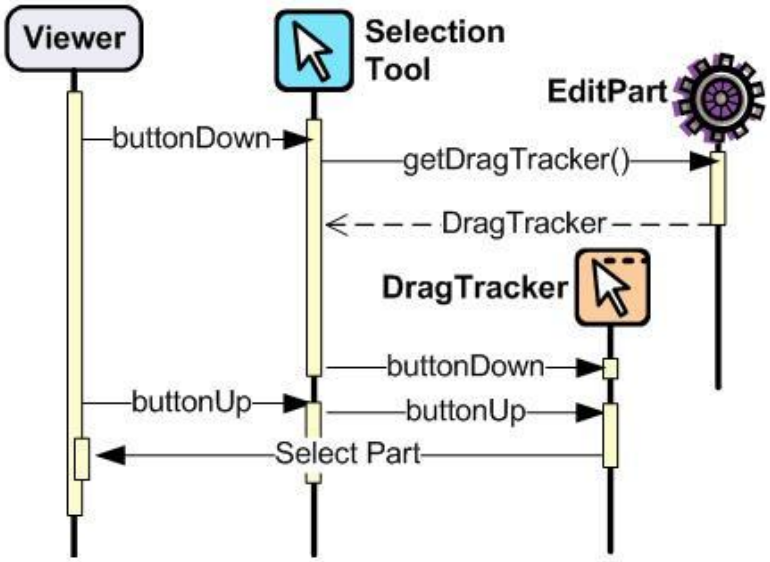
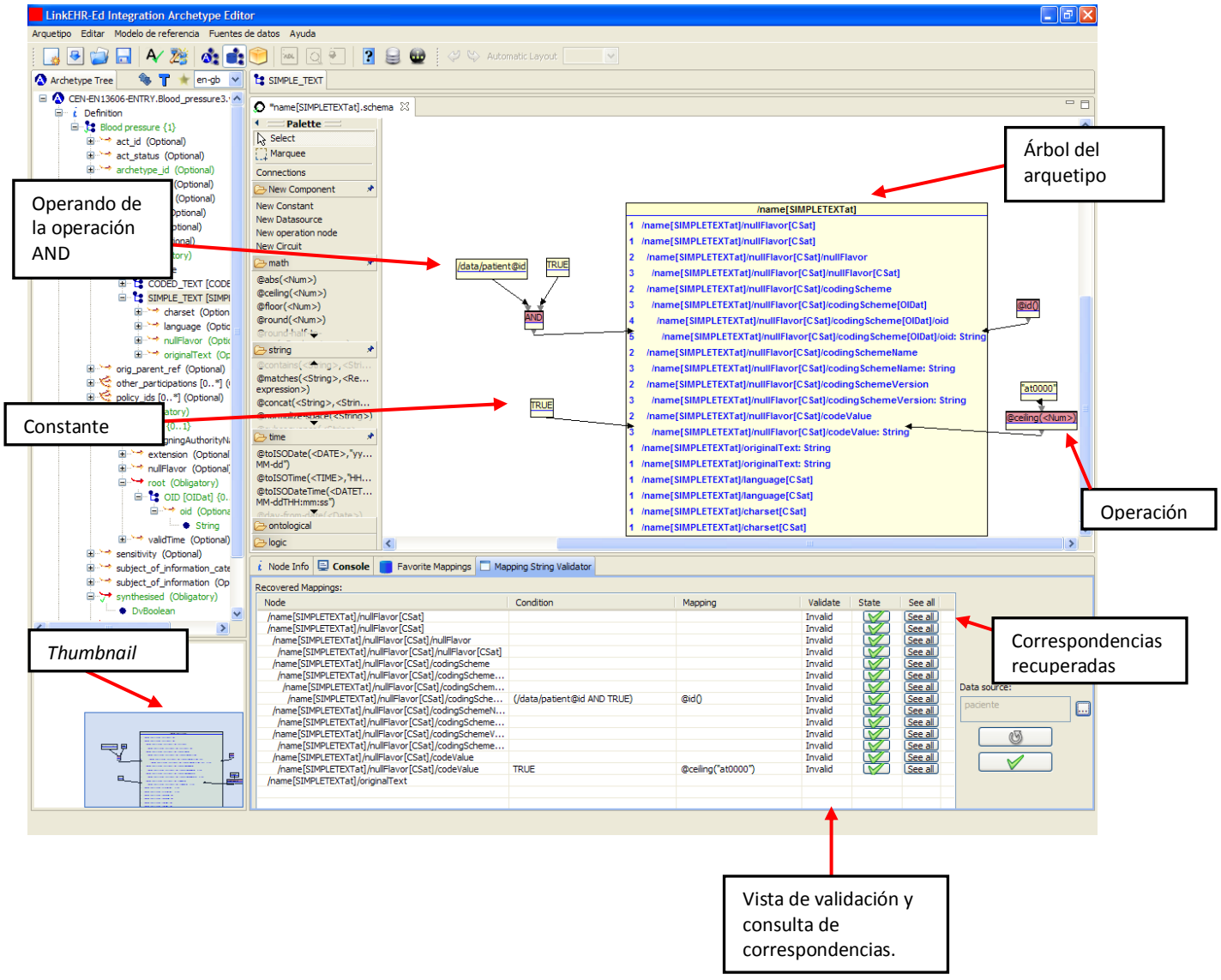


Figura 44: Diagrama de secuencia que describe el funcionamiento del *DragTracker* (13).

7. Implementación

En el presente capítulo se describe el proceso de implementación llevado a cabo para el desarrollo de GMM. Además se presenta a grandes rasgos la interacción entre las principales clases del proyecto para conseguir el comportamiento deseado para el editor.

El resultado al que se quiere llegar tras la implementación es el mostrado en la figura a continuación.



7.1. Estrategia seguida para la implementación

Puesto que la arquitectura de la aplicación es Modelo-Vista-Controlador, para poder depurar cada una de las funcionalidades desarrolladas se hacía necesario seguir un proceso de desarrollo incremental a partir de un producto esencial mínimo. De esta forma se puede comprobar el correcto funcionamiento de cada una de las opciones de la aplicación conforme se acaba su desarrollo. Se ha seguido una aproximación al modelo de desarrollo incremental.

7.2. Desarrollo de los componentes que integran el diagrama

Para conseguir la funcionalidad completa de un componente es necesario el desarrollo de su modelo, políticas, comandos y adhesión al esquema. Cada uno de los componentes se ha desarrollado de forma idéntica y vertical a la arquitectura desarrollando su parte del modelo, de la vista y del controlador.

Un ejemplo de cómo se han desarrollado los componentes se encuentra a continuación. En la figura que sigue se observa cada una de las clases que intervienen en la creación de una tabla. Una tabla es el objeto gráfico que puede tomar un valor cualquiera constante. Así, se desarrolló primero el objeto que representa a este componente en el modelo, posteriormente la política adecuada, la figura, el *part* que relaciona la figura con el modelo y, por último, el comando que permite ejecutar la acción para añadir una tabla al diagrama.

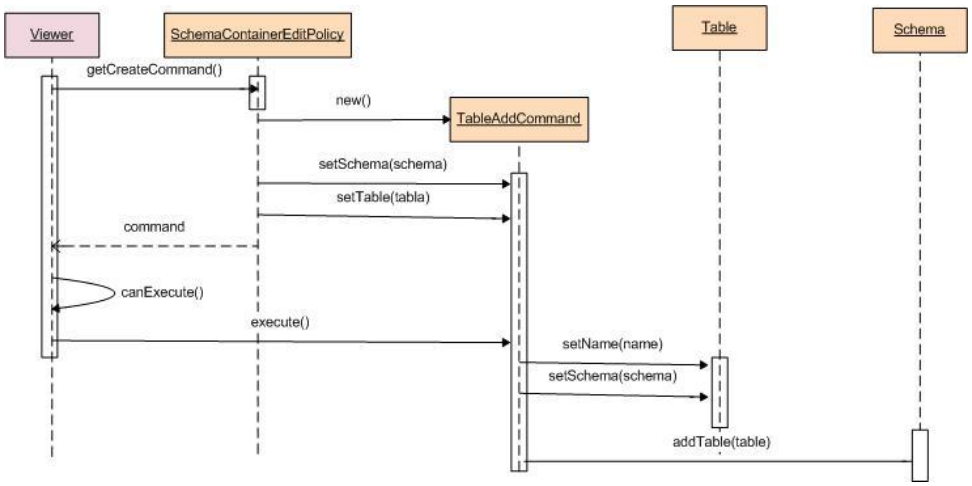


Figura 45: Adición de una tabla al esquema.

7.3. Validación de una Correspondencia

La opción de validación de una correspondencia supone uno de los puntos críticos en el programa ya que es el punto de unión entre GMM y el núcleo de LinkEHR-Ed.

El proceso para la validación/salvado de una correspondencia es el siguiente:

1. Recuperación de la cadena equivalente al grafo en el diagrama.

2. Construcción del objeto *MappingManager*. Este objeto es el que comunica la interfaz con el núcleo de LinkEHR-Ed cuando se trata de trabajar con correspondencias.

3. Llamar a la instancia del *MappingManager* para consultar la validez de la correspondencia.

4. Si la correspondencia es válida, y se desea guardarla, se recupera la función de correspondencia en uso del *AttributeMapping* con el que se está trabajando y se le añade un nuevo *FunctionMapping* correspondiente a la cadena de correspondencia recuperada.

La siguiente figura muestra el diagrama de secuencia para la validación de una correspondencia más detalladamente.

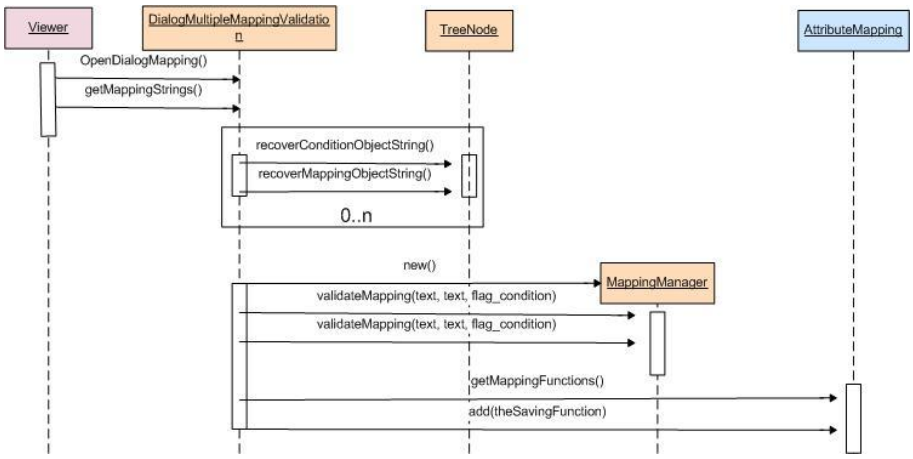


Figura 46: Validación de una correspondencia.

7.4. Inicialización del Editor

La inicialización del editor puede llevarse a cabo en dos circunstancias bien diferenciadas:

- El editor ya ha sido abierto para la correspondencia que se exige mostrar y se recupera el archivo que había sido salvado por el usuario. Esta opción permite continuar un trabajo que haya sido dejado sin salvar como una nueva correspondencia pero se ha guardado como instancia del editor. Los pasos para recuperar un editor que había sido previamente salvado son:

    1. Crear un flujo de bytes a partir del fichero salvado.
    2. Leer el flujo de bytes y usarlo para alimentar al editor.
    3. Crear el grafo a partir del flujo de bytes recuperado.
    4. Crear un *ContentCreator* para apoyo a las tareas de validación.

- No existía un archivo relacionado con la correspondencia que se desea editar y se debe construir un grafo en consonancia a la correspondencia referida.

    1. Crear un *ContentCreator* para la construcción del grafo a partir de la correspondencia.
    2. El *ContentCreator* recorre la estructura de datos de la correspondencia construyendo los objetos gráficos que corresponda.
    3. Recuperar y asociar al editor el esquema devuelto por el *ContentCreator*.

El diagrama de secuencia presenta los pasos que se dan para la inicialización de un editor en cada una de las circunstancias descritas.

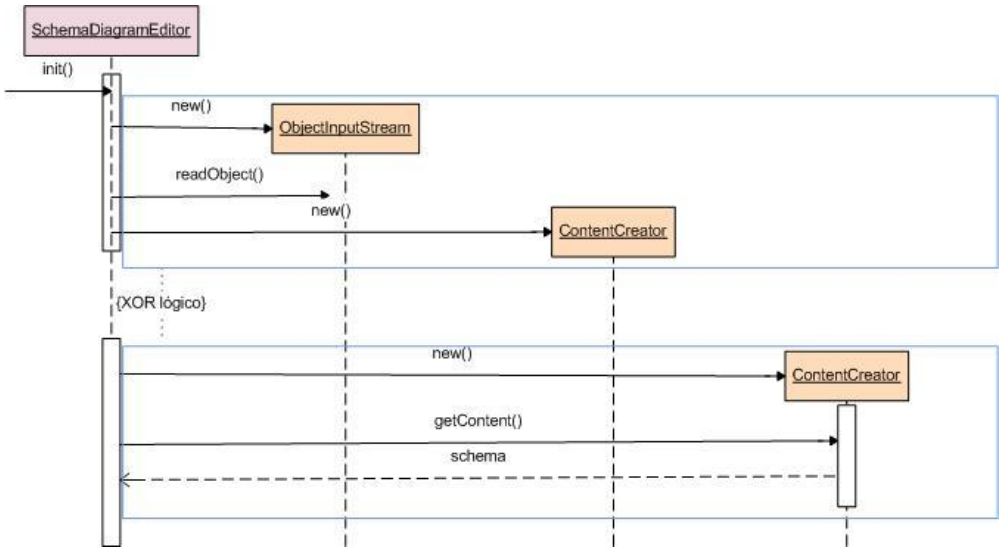


Figura 47: Inicialización del editor.

7.5. Construcción de una conexión

El proceso de conectar los diferentes componentes es la clave para la elaboración de una correspondencia. Mediante la conexión de operadores (fuentes de datos, constantes u operaciones previas) con las operaciones, conseguimos que a la fuente de datos se le apliquen las transformaciones requeridas.

Para la correcta conexión de dos componentes se llevan a cabo las siguientes interacciones:

1. La vista reclama la construcción de un comando para la creación de conexiones.
2. La política correspondiente crea el comando necesario y le aporta el origen de la conexión.
3. Si la conexión entre los componentes solicitados está permitida se llama a la ejecución del comando.
4. Se crea el objeto *Relationship* y se le asignan las instancias que corresponden a origen y destino en el nivel de modelo.

El diagrama muestra cómo interactúan el *Viewer*, la política y el comando para la creación de una relación entre dos componentes.

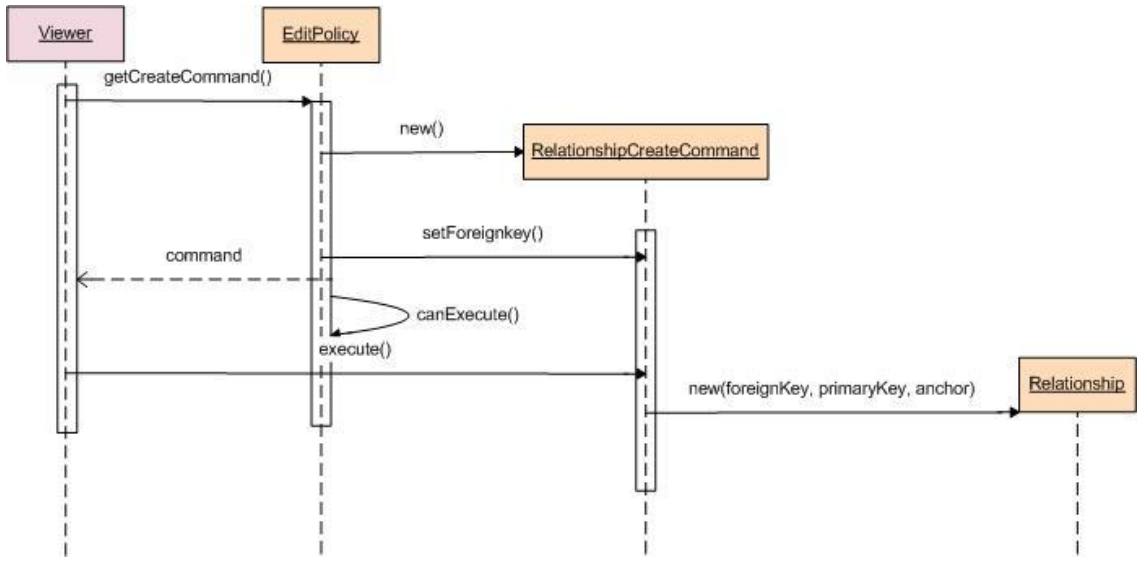


Figura 48: Creación de una relación.

7.6. Front-End de GMM

Graphical Mapping Manager ha sido desarrollado bajo en el *framework* GEF que se apoya sobre la librería draw2d del IDE Eclipse. Todo lo que es visible al usuario se corresponde con los componentes que aporta draw2d, con excepción de los diálogos y menús, los cuales se basan en la librería SWT (17).

7.6.1. Inicializando GMM

El lanzamiento del editor se realiza desde la pantalla principal de LinkEHR-Ed accionando la opción correspondiente a GMM de la barra de herramientas. Para la correcta apertura del editor, será necesario haber seleccionado previamente uno de los nodos del arquetipo.

La figura muestra como el usuario acciona el botón de la barra de herramientas para lanzar GMM. Previamente ha seleccionado el nodo del arquetipo del cual obtendrá el subárbol que cuelga de él.

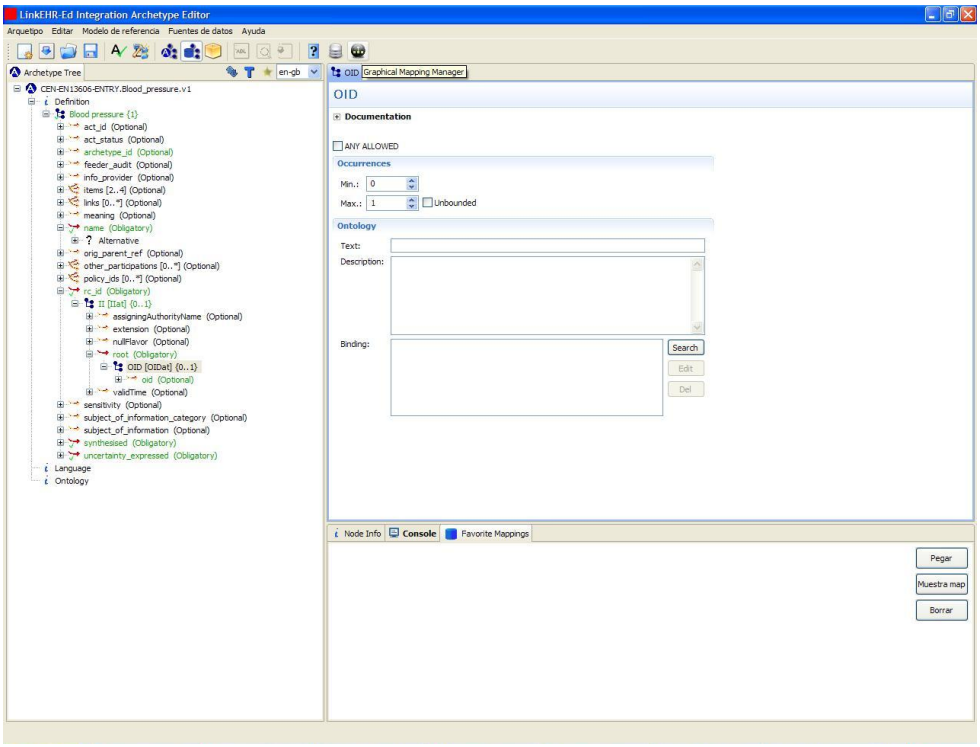


Figura 49: Árbol de correspondencia.

7.6.2. Arrastrando una Operación al Diagrama

Para incluir una operación en el editor basta con arrastrarla desde la paleta de herramientas hasta el espacio de edición. La aplicación se encargará de construir la figura adecuada para el tipo de operación seleccionado.

En la captura de pantalla se muestra la operación lógica OR que acaba de ser arrastrada al diagrama.

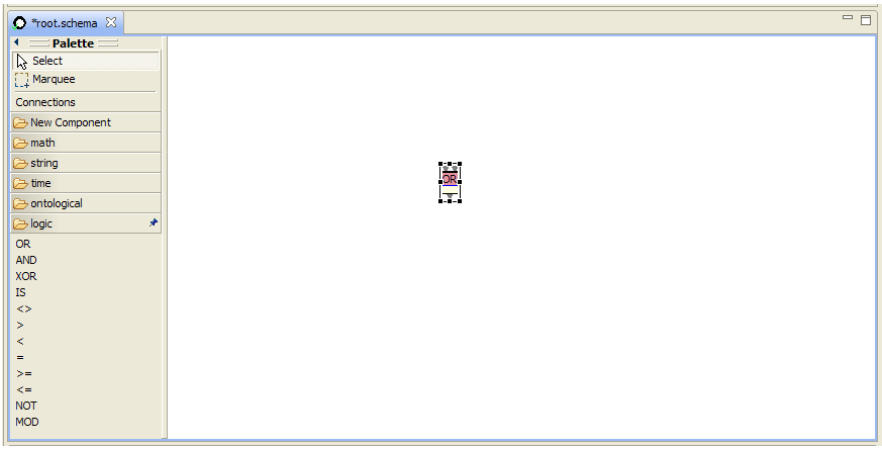


Figura 50: Arrastrando una operación al diagrama.

7.6.3. Insertando Fuente de Datos

El proceso de inclusión de una fuente de datos es similar al de la operación. La diferencia estriba en que antes de crearse la figura deseada en el editor, será necesario seleccionar el nodo que queremos que se incluya de un diálogo donde aparece la estructura de la fuente de datos.

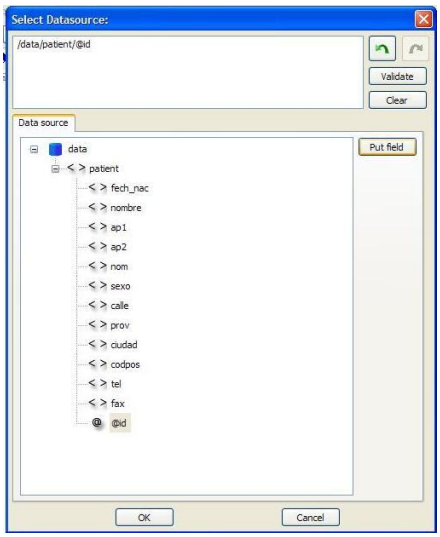


Figura 51: Seleccionando una fuente de datos del diagrama.

Tras realizar la selección del nodo deseado, la aplicación creará la figura correspondiente a esa fuente de datos.

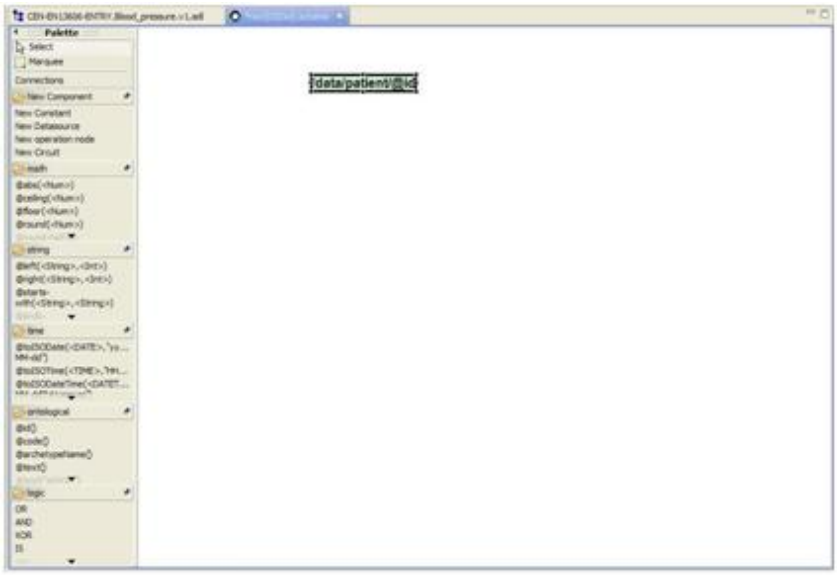


Figura 52: Creación de una fuente de datos.

7.6.4. Editando Tabla

Como se muestra en la figura los componentes del grafo para la correspondencia tienen la posibilidad de edición directa haciendo un clic sobre ellos. Solo se deberá introducir el nuevo valor y presionar *enter* para fijar el nuevo valor tanto en la figura como en el modelo.

Figura 53: Edición de una tabla.

7.6.5. Conectando Elementos

La conexión de elementos se realiza seleccionando de la paleta la opción conexión y conectando del nodo fuente al destino. Se debe notar que la conexión aparecerá como prohibida hasta que nos aproximemos a un punto de anclaje válido. Si intentamos conectar un elemento a otro que tenga su cupo de conexiones completo, o para el que el origen no sea válido, el usuario advertirá que el símbolo de prohibición no desaparece aunque se aproxime a un punto de anclaje.

Al finalizar correctamente la creación de una conexión ésta se verá como en la figura a continuación. Además el modelo de cada una de las figuras habrá almacenado una conexión fuente y un destino respectivamente.

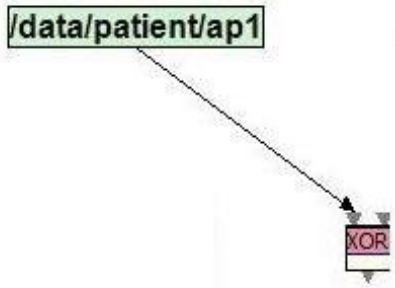


Figura 54: Conexión de elementos.

7.6.6. Validación de Correspondencias

La acción de salvar correspondencias puede llevarse a cabo una vez se han recuperado las correspondencias del grafo y se tienen en la vista de validación de correspondencias en forma de cadena.

Pulsando sobre el botón  se valida la correspondencia de la fila. La validación se hace tanto sobre el filtro como sobre la función. Si la correspondencia es correcta se le dará al usuario la posibilidad de salvarla mediante el menú que se muestra en la captura de pantalla.

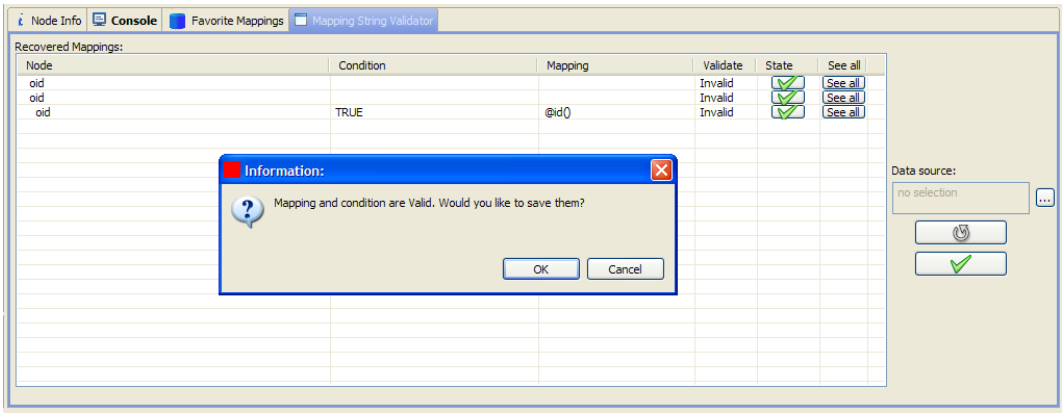


Figura 55: Menú para salvar una correspondencia.

7.6.7. Consulta de Todas las Correspondencias de un Nodo

Pulsando sobre el botón *See All* del diálogo de la figura anterior se muestran todas las correspondencias existentes para un determinado nodo. Además para cada una de ellas se dará la opción de mostrar la correspondencia en forma gráfica activando el botón *Show* de la captura siguiente.

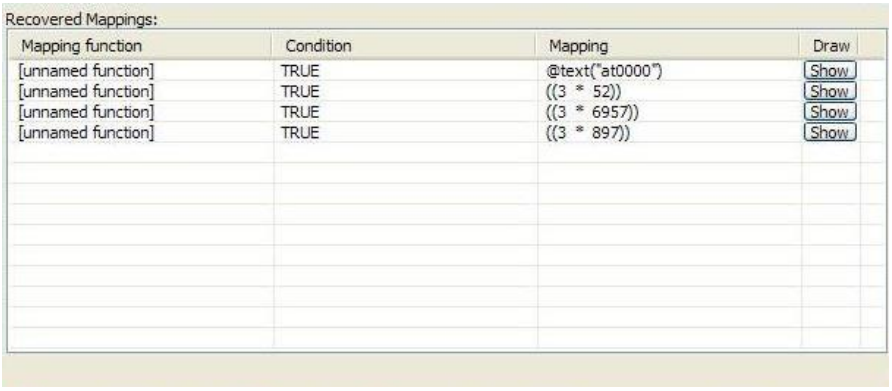


Figura 56: Vista de todas las correspondencias de un nodo.

7.6.8. Mostrar Menú del Editor

Ciertas acciones están asociadas al menú contextual que se puede visualizar pulsando el botón derecho sobre el lienzo del editor. Las opciones que este menú contiene son:

- Deshacer la última acción realizada sobre el editor.
- Rehacer una acción deshecha.
- Pasar de modo automático de *layout* a manual y viceversa.
- Lanzar el diálogo que, análogamente a la vista de validación, permite la recuperación de correspondencias en forma de cadena, así como la validación y salvado de estas.

Estas opciones se corresponden con cada una de las que aparecen en la figura siguiente:

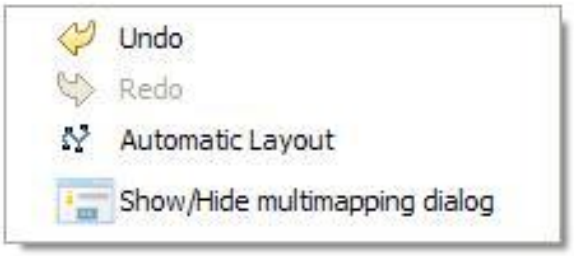


Figura 57: Menú contextual.

7.6.9. Vista General

En la siguiente figura se muestra una vista general del editor con diversas correspondencias asociadas a diferentes nodos del arquetipo. A la izquierda puede observarse la paleta con la mayoría de las opciones a la vista. Abajo se observa la vista para la validación de las correspondencias. Ésta posee una tabla que permite visualizar los pares filtro/función de cada correspondencia.

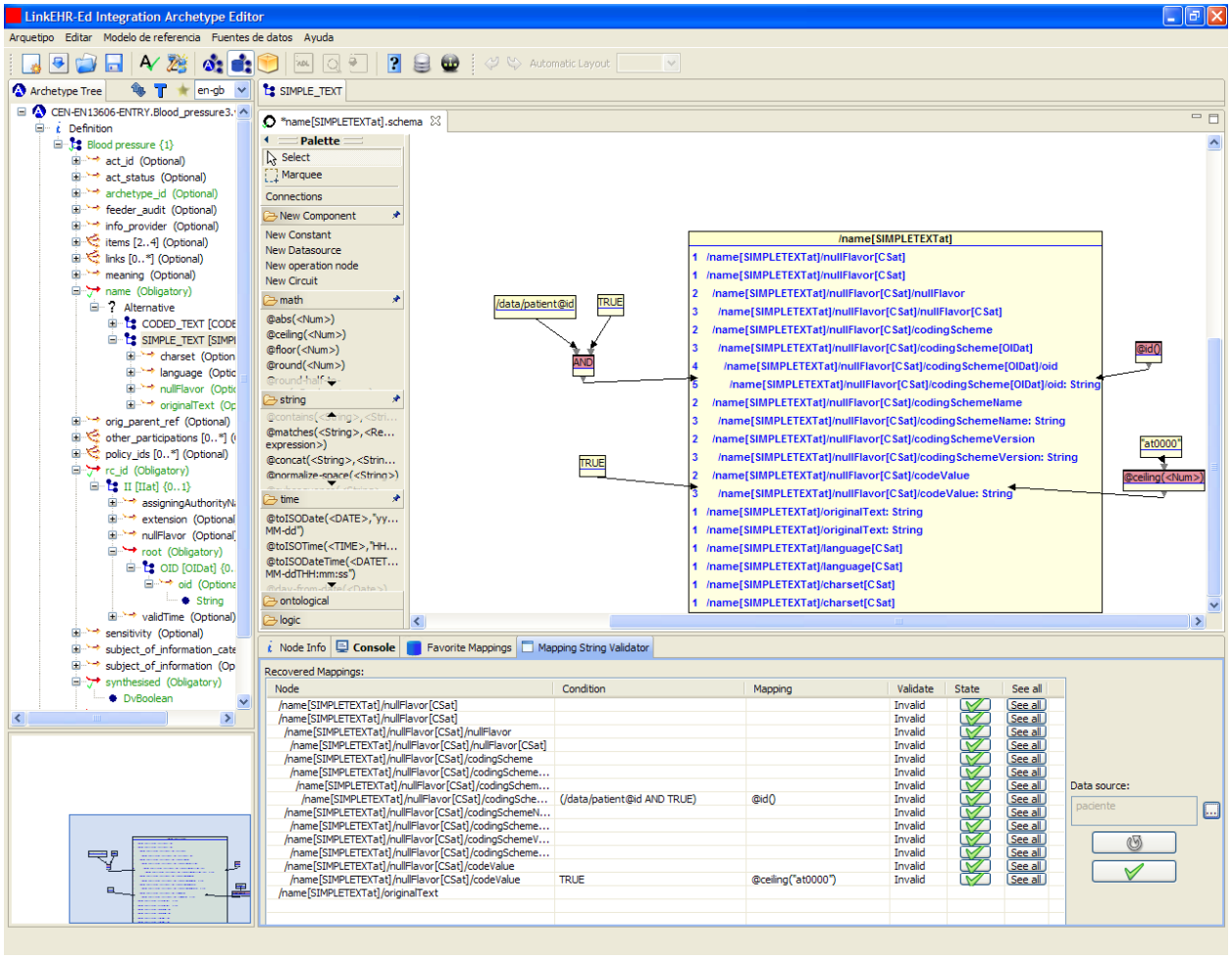


Figura 58: Vista general del editor.

8. Comparación con sistemas similares

Altova Map Force

Esta es actualmente la herramienta líder del mercado en lo que a mapeo e integración de datos se refiere. Con tan solo enlazar las fuentes de datos con las operaciones y estas con los destinos se consigue establecer correspondencias. La interfaz permite un fácil aprendizaje ya que el uso de los componentes es muy sencillo.

Una vez establecidas las correspondencias, la herramienta es capaz de generar un programa en *XQuery* que extraiga los datos de las fuentes y los deposite en el destino especificado tras aplicarle las operaciones pertinentes.

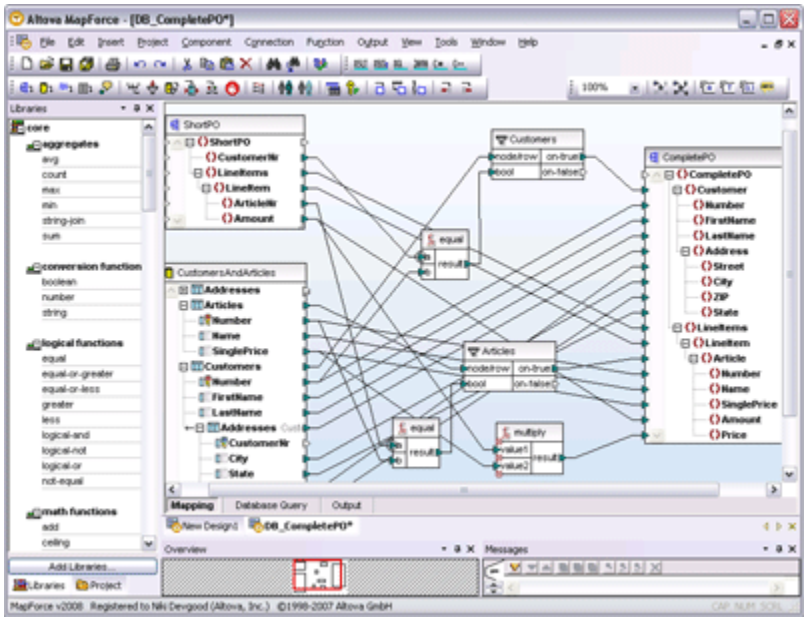


Figura 59: Altova Map Force.

Aunque Map Force es una de las mejores herramientas disponibles, su principal desventaja visual es que la gestión de correspondencias complejas que requieran muchos enlaces genera una gran cantidad de líneas. Esto puede confundir al usuario y no permitirle entender la secuencia de funciones que se está aplicando. Se puede observar en la figura anterior como, para una correspondencia relativamente sencilla, el lienzo aparece copado de conexiones. Además Map Force presenta la limitación de que es solo una aplicación y no puede actuar como servidor. Esto implica que el transporte de información deberá basarse en otra tecnología quedando Map Force únicamente para el mapeo de los datos.

Rational Data Architect

Rational Data Architect (18) es un producto de IBM diseñado para el modelado y la integración de datos de la empresa. Sus principales características son:

Modelado de Datos Robusto: permite a los arquitectos y diseñadores crear modelos lógicos, físicos y de dominio fácilmente.

Mapping: ayuda a revelar y definir las relaciones entre fuentes de datos y objetivos.

Análisis de Conformidad de Modelo y Base de Datos: se ejecuta contra modelos o bases de datos existentes y comprueba la conformidad a los patrones, normas y reglas de la empresa.

Information Integration Design: simplifica el diseño, creación y despliegue de bases de datos federadas.

Comparar y Sincronizar: permite que los arquitectos comparen modelos con modelos, modelos con bases de datos y bases de datos con bases de datos y actualicen posteriormente bases de datos existentes para asegurar la consistencia.

Análisis de Impacto: ayuda a identificar el impacto de un cambio antes de que el cambio sea implementado. El análisis de impacto listará las dependencias sobre un elemento específico. Los resultados son representados visualmente y también desplegados en formato sencillo.

La Integración del Ciclo de vida: ayuda a integrar a los usuarios con las otras facetas del ciclo de vida. Los requerimientos almacenados y administrados en ***Rational RequisitePro*** pueden ser accedidos o asociados a los correspondientes elementos de modelado y sincronizados con reglas seleccionables por el usuario.

Como se observa en las características, Rational Data Architect es un excelente producto para la integración de las distintas fuentes de datos de una organización. Como crítica cabría decir que es una herramienta de integración con ciertas limitaciones a la hora de integrar orígenes de datos que no sean de IBM. Un ejemplo de ello han sido los problemas que presenta a la hora de conectar con sistemas que no sean DB2 o el soporte limitado a MySQL.

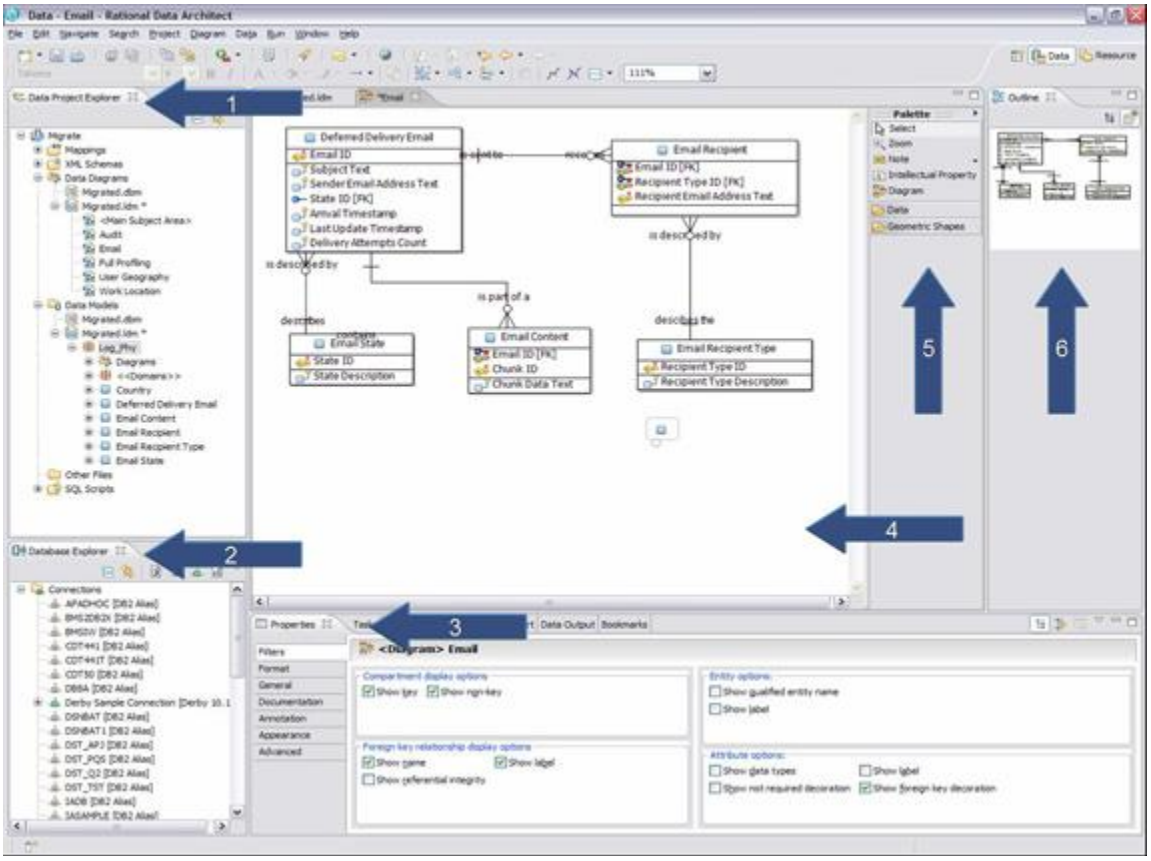


Figura 60: Rational Data Architect.

Microsoft Biz Talk

Esta herramienta de Microsoft provee de interconectividad entre las aplicaciones de la organización. A su vez aporta mensajería, un motor de reglas, conectividad mediante ***E****lectronic* ***D****ata* ***I****nterchange*, *Business Activity Monitoring* y conectividad RFID.

Biz Talk permite la integración de sistemas tan dispares como pueda ser un ERP o un sistema de identificación basado en RFID.

De esa forma permite el uso integrado de las tecnologías existentes en la organización sin llevar a cabo cambios integrales e instalación de sistemas de grandes dimensiones. Como se observa en la figura los principales departamentos y sistemas de la organización pueden ser integrados mediante un servidor BizTalk. Al ser un servidor puede gestionar el transporte y mapeo de la información por sí mismo.

Su principal desventaja según la comparativa de *scruffylookingcatherder* (19) es que Biz Talk precisa de un servidor de bases de datos SQL Server 2000 para el almacenamiento de sus mapeos no siendo útil otra versión de éste.

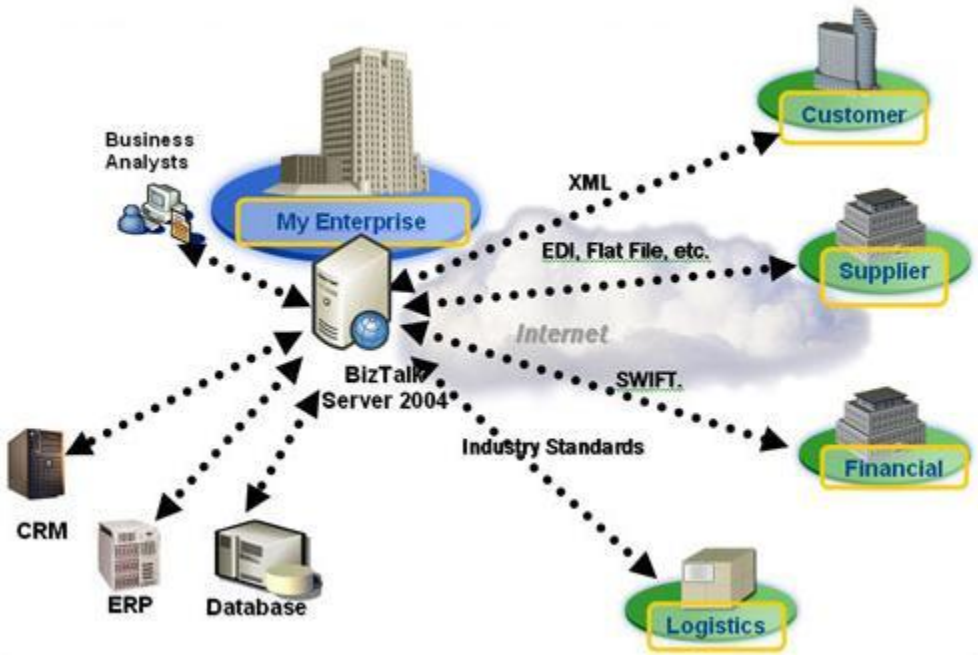


Figura 61: Microsoft Biz Talk.

Stylus Studio

Esta aplicación se centra en XML como forma de intercomunicar diferentes sistemas. Provee de posibilidades como establecer correspondencias de XML a XML, BBDD a XML, HTML a XML, EDI a XML etc. Su principal ventaja es que es uno de los mejores motores para convertir de EDI a XML, ello hace que tenga un completo soporte a HL7. En la figura se muestra el proceso de transformación de EDI a XML.

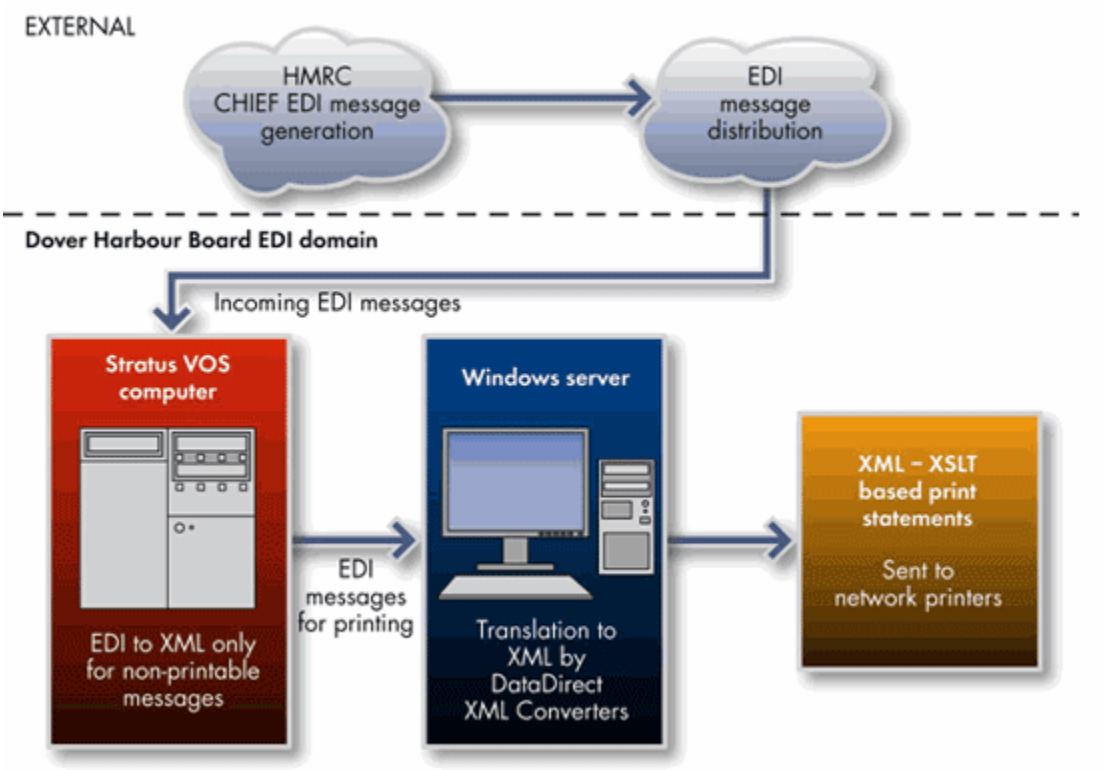


Figura 62: Stylus Studio.

Comparativa

Los sistemas antes mencionados son superiores al GMM de LinkEHR-Ed en versatilidad por la gran variedad de orígenes de datos que son capaces de integrar. No obstante, la innovación de GMM consiste en la posibilidad de definir correspondencias entre fuentes de datos relacionales o XML con arquetipos.

Este sofisticado tipo de correspondencias no está soportado por ninguna aplicación comercial. Esto se debe a la imposibilidad de representar un arquetipo en *XML-Schema.* Los arquetipos pueden contener nodos hermanos con la misma etiqueta, de hecho es algo presente en la inmensa mayoría de arquetipos. XML-Schema no permite que los nodos hermanos puedan tener la misma etiqueta (atribución de partícula única o *Unique Particle Attribution*), para

facilitar su procesado. Por tanto, la inmensa mayoría de arquetipos no se pueden representar por medio de XML-Schema, ni obviamente por un DTD que son aún más restrictivos que los XML-Schema. Así se convierte GMM en la única herramienta capaz de asociar los diferentes orígenes de datos a los nodos de un arquetipo, pudiendo integrar sistemas mediante estas estructuras. A su vez cabe hacer mención a otra problemática que las herramientas anteriores no son capaces de resolver. Ésta está relacionada con la definición de directivas para las transformaciones estructurales entre fuente y destino. Este problema queda resuelto en LinkEHR-Ed mediante el método mencionado en el apartado 3.3.1.

Finalmente, como ventaja en representación visual, aporta la capacidad de tener una instancia del editor diferente para cada mapeo en curso evitando el problema de los diagramas sobrecargados.

9. Conclusiones y trabajos futuros

Los sistemas de información sanitarios se encuentran en una constante evolución hacia la estabilidad y madurez de las tecnologías. El reto de una aplicación como LinkEHR-Ed es la integración de los diferentes sistemas existentes y conseguir la interoperabilidad entre ellos de la forma menos traumática posible. Al mismo tiempo, se debe facilitar la utilización del uso de herramientas de integración al usuario, minimizando la formación necesaria para poder explotar una determinada tecnología.

El objetivo del presente proyecto ha sido lograr el desarrollo de un editor que facilite la edición de correspondencias de fuentes de datos en el motor de integración LinkEHR-Ed.

El desarrollo de este editor se ha llevado a cabo siguiendo un proceso iterativo. En cada fase de este proceso se ha desarrollado una de las funcionalidades de forma vertical y completa hasta conseguir el conjunto de funciones que permitan, mediante su combinación, la edición de la mayoría de correspondencias. Como la evolución de los sistemas hace necesaria la incorporación continua de nuevas funciones de correspondencia, el editor ha sido diseñado de forma que la inclusión de una nueva función se lleve a cabo sin necesidad de ser codificada. Para ello el editor lee las funciones y las propiedades de éstas de un fichero XML, en el cual se describen todas sus características. De esa forma, basta con añadir la descripción de la función al fichero mencionado para que esté disponible en el editor.

El desarrollo de GMM no acaba con el cierre de este proyecto. Nuevas funcionalidades, como la visualización de múltiples correspondencias de un solo nodo o mejoras en la representación visual deberán ser llevadas a cabo. Por otro lado las tecnologías utilizadas para el desarrollo de la aplicación y otras ligadas a ellas pueden aportar una gran capacidad de representación para campos como el de las terminologías médicas. La tediosa navegación que se hace sobre éstas podría hacerse de una forma mucho más eficiente con la explotación de librerías gráficas de altas prestaciones.

10. Publicaciones relacionadas

Jose A. Maldonado, David Moner, Diego Boscá,, Luis Marco, Jesualdo T. Fernández-Breis[b], Ernesto Reig, Montserrat Robles. *LinkEHR-Ed: a tool for the description and normalization of legacy clinical data.* EFMI Special Topic Conference 2010. Reykjavik, Islandia. Véase anexo I.

ANEXO I

LinkEHR-Ed: a tool for the description and normalization of legacy clinical data

Jose A. MALDONADO[a,2], David MONER[a], Diego BOSCA[a], Luis MARCO[a], Jesualdo T. Fernández-Breis[b], Ernesto REIG[a] and Montserrat ROBLES[a]
[a] Instituto ITACA, Universidad Politécnica de Valencia, Spain
[b] Departamento de Informática y Sistemas, Facultad de Informática, Universidad de Murcia, Spain

Abstract. In this paper, we present the mapping capabilities of LinkEHR-Ed, a visual tool for clinical data description and normalization based on archetypes. It provides functionalities that simplify the process of setting up integration or data exchange engines. Key contributions of LinkEHR-Ed are the development of a powerful archetype editing framework capable of handling multiple electronic health record architectures, a visual environment for defining high-level non-procedural mappings between archetypes and legacy data structures and algorithms for automatically generating XQuery scripts from the high-level mappings. The execution of the XQuery scripts yields as a result XML documents that are compliant with the underlying electronic health record data architecture and at the same time satisfy the constraints imposed by archetypes.

Keywords. Electronic Health Records, Normalization, Mappings, Data Exchange, Archetypes.

Introduction

Health care is a sector where the need of sharing information is the norm rather than the exception. However, the health data of one patient is usually scattered among different Electronic Health Record (EHR) systems. This leads to distributed and heterogeneous data resources creating a large gap between the potential and actual value of the information content of EHR systems. Closing this gap by making efficient use of the health data held by these systems, could improve significantly patient care, clinical efficiency, patient safety and empower research activities.

Data normalization is a prerequisite to achieve semantic interoperability in any domain. This is even more important in the healthcare sector due to the special sensitivity of medical data. Data exchange must be done in a meaningful way, avoiding all possibility of misunderstanding or misinterpretation. As a consequence any project involving the sharing or communication of EHR must face the challenge of how to model the data to be communicated. The first and basic requirement is the definition of a common information architecture for communicating EHR extracts. Considerable effort has been invested over the years on the definition of EHR architectures [1]. A novel approach is the so-called two-level model (also known as dual model). This new approach tries to overcome the problems caused by the complexity and the continuous evolution of the health domain. Two models are distinguished: the reference model and archetypes. The Reference Model contains the basic and stable entities for representing any entry in an EHR. The second model comprises the set of volatile domain and application-specific concepts such as blood pressure, discharge report, etc. These concepts are modeled as archetypes which are formal definitions of clinical concepts defined in the form of constraints on the generic data structures of the reference model. Therefore, both models are highly related.

[2] Corresponding Author: Jose Alberto Maldonado, Instituto ITACA, Edificio 8G, Universidad Politécnica de Valencia, Camino de Vera s/n 46022, Valencia, Spain. E-mail: jamaldo@upv.es

In this paper we present LinkEHR-Ed, a visual tool that aims at facilitating the semantic description of legacy EHR in the form of archetypes and the normalization of their clinical content. LinkEHR-Ed contains two main modules, i) a multi-reference model archetype editor and ii) a mapping generation module that automatically creates XQuery data transformation programs from a set of correspondences between the elements of the archetype and the source schema. This paper focuses on the mapping module. Nevertheless, some details will be given regarding the archetype editing framework. The use of LinkEHR in an EHR exchange scenario starts with the definition of the information to be shared in the form of archetypes by a domain expert. Next and using a graphical interface, the data engineer specifies high-level correspondences between the elements of both schemas (archetype and legacy data source). Finally, LinkEHR-Ed compiles the set of values mapping in an XQuery script. The execution of the resulting script on a data source instance generates an XML document that satisfies the archetype constraints and is compliant with the underlying reference model. Our tool is designed to ease the construction of EHR exchange systems. For instance, the resulting XQuery script can be used by integration engines to normalize and semantically describe the clinical data that they offer.

Dual Model EHR Architectures

ISO/CEN EN13606 and openEHR architectures are based on the dual model approach to describe the structure and semantics of health data. The dual model methodology distinguishes a reference model and archetypes [2]. In a broad sense, a reference model is an abstract representation of the entities and relationships of a domain which is designed to provide a basis for the development of more concrete models and implementations. The generality of the reference model is complemented by the particularity of archetypes. Archetypes are detailed and domain-specific definitions of clinical concepts in the form of structured and constrained combinations of the entities of the reference model.

Only those classes of the reference model that define logical building blocks of EHRs can be used as basis for defining archetypes. We will call them business concepts. For instance ISO/CEN EN13606 defines six business concepts, namely: Folder, Composition, Section, Entry, Cluster and Element. What is important here is that for each domain concept, a definition can be developed in terms of constraints on structure, types, values, and behaviors of business concepts. Archetypes may be specialized: an archetype is considered a specialization of another archetype if it specifies that archetype as its parent, and only makes changes to its definition such that its constraints are narrower than those of the parent. ADL (Archetype Definition Language) [3] is a formal language developed by the openEHR consortium for expressing textually archetypes that has also been adopted by CEN EN13606.

Mapping definition in LinkEHR-Ed

LinkEHR-Ed (http://www.linkehr.com) is a visual tool implemented in Java that uses archetypes as a means to achieve the normalization of existing health data. The main objectives are twofold. Firstly, in the context of data integration, the use of archetypes as a semantic layer over the data repositories, whose contents need to be integrated, described and exchanged, associating them with formal semantics. Secondly, the use of archetypes for making public existing clinical information in the form of normalized/standardized EHR extracts. The archetype mapping process is divided into four main steps in LinkEHR-Ed. The first step deals with the importation of reference models. A new reference model expressed as a W3C XML Schema can be imported at any time. The second step is the actual archetype editing process, for this purpose a visual editor is provided. Step 3 is about mapping specification between archetypes and data sources. In step 4, an XQuery expression that encodes the mapping specification is automatically created. The execution of the XQuery expression over a set of data sources yields a XML instance that satisfies the constraints imposed by the archetype and at the same time is compliant with the underlying reference model. Therefore, LinkEHR-Ed helps data engineers to write possibly long and complex programs to standardize data. In summary, LinkEHR-Ed combines the formal representation of knowledge of a health domain expert in the form of archetypes (steps 1 and 2), with mapping information to clinical data sources for the standardization and semantic description of existing information (steps 3 and 4).

Archetype Editing

The archetype editing process in LinkEHR-Ed is based on a clear formalization of the archetype model. The formalization is based on types over tree-structured data with labeled nodes (i.e. XML data) and a subsumption

relation is used to model the specialization of archetypes and the relationship between archetypes and reference models. For a detailed explanation we refer to [4].
In LinkEHR-Ed a new reference model expressed as W3C XML Schema can be imported at any time. Users must indicate the set of classes of the reference model that define logical building blocks of EHRs, i.e. the set of business concepts. Obviously, this step only needs to be performed once for each reference model. One of the main features of LinkEHR-Ed is that the set of business concepts are also represented as archetypes, we call them business archetypes. Note that business archetypes are the most general archetypes that can be defined for a reference model and therefore any archetype must be a specialization of one of them.
Business archetypes allow us to apply the same logic either when a new archetype is defined from scratch or when an existing one is specialized to create a more refined medical concept. As consequence, only the rules specified in the archetype model to control the archetype creation need to be directly implemented in the tool. This allows the definition of archetypes based on different standards. Four reference models have been tested successfully, namely ISO/CEN EN13606, OpenEHR, CDA, and CCR. To the authors' knowledge, LinkEHR-Ed is the only editor capable of handling ISO/CEN EN13606.

Mapping of Archetype to data sources

Existing clinical data must be transformed to meet the data structures and constraints defined by reference models and archetypes. Therefore mapping specifications must define how to create archetype instances from the content of the data sources. In LinkEHR-Ed, an archetype with a mapping specification to a data source is called an integration archetype. We face a problem known in the literature as the data exchange problem [5]. Data exchange at the schema level requires an explicit representation of how the source schema (legacy data schema) and target schema (archetypes) are related to each other; these explicit representations are called mappings. The effort required to create and manage such mappings is considerable. It involves writing and managing complex data transformations programs. The common case is to write intricate custom and non-reusable software to perform the required transformations. This is even more complex with archetypes, since they are defined to model arbitrary complex domain concepts without any consideration regarding the potential internal architecture or database design of EHR systems. As a consequence, complex and expressive mapping specifications are necessary. It is not only necessary to specify value couplings between atomic entities but also structural mappings, i.e. how the data structures of source and target schemas are related, due to the potential low similarity between archetypes and clinical data sources. In the health care domain very few generic EHR data transformation efforts exist. Many commercial mapping tools are capable of processing XML schemas but they cannot handle archetypes.

LinkEHR-Ed is designed to map hierarchical data; the source schema must be expressed in W3C XML Schema and the target schema (archetypes) in ADL. LinkEHR-Ed can also handle source relational schemas, as long as they are converted in a canonical way into a W3C XML Schema. For this purpose, it comes with a visual tool for handling relational schemas that can also generate a set of Hibernate mapping definitions that construct such canonical XML views. During the mapping definition from relational sources users do not have to interact with the canonical XML view, instead a visual representation in the form of tables and relationships is provided.

In ADL only those entities (classes and attributes) of the reference model that are actually constrained need to appear in the archetype definition. This rule poses a difficulty when an archetype is mapped to a data source. It may be necessary to define a mapping for an unconstrained attribute and therefore it is not present in the archetype definition. Thus, for the sake of mapping we are forced to temporally complete the archetype definition with entities from the underlying business archetype. In LinkEHR-Ed, we have defined a merge function that takes an archetype and a business archetype as input and outputs what we call a comprehensive archetype, which contains all the entities defined in the latter but at the same time preserves all the constraints imposed by the former. Put in other words, comprehensive archetypes include all the explicit constraints (those defined by the archetype to be specialized or mapped) and all the implicit ones (those defined by the business archetype) that data instances must satisfy. Figure 1 shows an example of comprehensive archetype. On the left-hand side the original ISO/CEN EN13606 archetype is depicted whereas on the right-hand side the corresponding comprehensive archetype is shown. Note that the comprehensive archetype contains all the constraints of the original archetype as well as all the unconstrained entities from the reference model class such as act_status, archetype_id, etc.

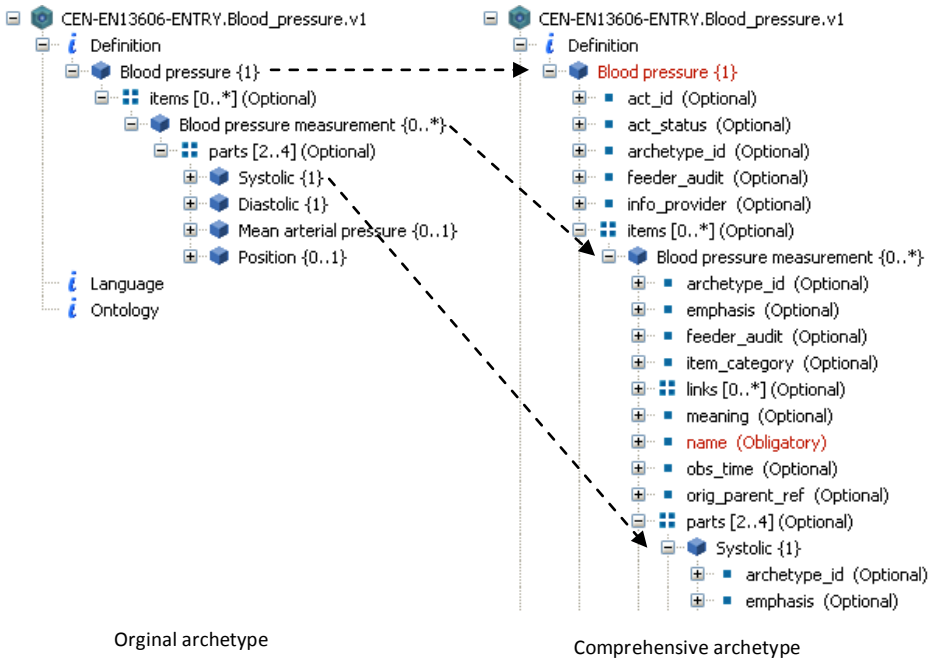


Figure 1. Example of comprehensive archetype.

LinkEHR-Ed can be considered as a high-level schema mapping tool. Users are responsible of defining a high-level non-procedural mapping specification. This high-level specification is defined by defining correspondences between entities of archetypes and source schemas. Two types of correspondences are supported, namely between atomic entities (leaf nodes) and between complex entities (inner nodes). The former are value correspondences that specify how to calculate atomic values whereas the latter are structural correspondence that may be used to control the generation and grouping of elements in the target.

Value correspondences are defined by a set of pairs, consisting of a transformation function that specifies how to calculate a value in the target from a set of source values, and a filter, indicating the conditions that source data must satisfy to be used in the transformation function. The simplest kind of transformation function is the identity function which copies a single source value into a target value. But quite often it is necessary to specify arbitrary complex functions which transform a set of source values into a target value. To achieve this, a wide range of transformation functions are supported such as type conversion, mathematical, logical, string, date and time, set value and ontological. When the filter is missing it is supposed that its value is true; hence it will be applied always. For instance, Figure 2 contains a simple correspondence transforming gender codes. It transforms the local gender code in the path: /patient/gender of the source XML EHR extract into a normalized code. Note that the order is relevant and only the first applicable function is used, consequently the last filter acts as a 'default' condition. Therefore, this correspondence should be interpreted as:

If (/patient/gender='M' OR /patient/gender='m') then 0
Else if (/patient/gender='W' OR /patient/gender='w') then 1
Else if (/patient/gender=0 OR /patient/gender=1) then /patient/gender
Else 9

Filter	Function
/patient/gender='M' OR /patient/gender='m'	0
/patient/gender='W' OR /patient/gender='w'	1
/patient/gender=0 OR /patient/gender=1	/patient/gender
true	9

Figure 2. An example of value correspondence.

Value correspondences are easy to specify but lack expressive power regarding grouping semantics, i.e. when target instances must be grouped and nested inside the same element. LinkEHR-Ed comes with a default groping semantics which is context-aware, in the sense that data with the same context (atomic elements at the upper levels) are grouped together. The context data depends on reference models and it is calculated on the fly for each archetype. For instance, in the case of ISO/CEN EN13606 data that share the same committed time, attester, etc. are grouped together.

On the other hand, structural correspondences [6] are defined by a set of source paths and a filter. They control the creation of target instances, in such a way that a new target instance is constructed for each set of source nodes addressed by the paths that satisfies the filter. It should be note that value correspondences are mandatory, at least one value correspondence must be define for every mandatory attribute in the target whereas structural correspondences are optional, if they are not present default grouping semantics is applied.

The abstract mapping specification is then compiled into an executable transformation script expressed using XQuery. The resulting transformation script takes as input an instance of the source EHR data and generates a XML document that is compliant both with the archetype and the underlying reference model. LinkEHR-Ed is a visual programming environment for the definition and management of abstract mappings and for the generation and validation against real data of the XQuery transformation scripts.

Evaluation and Conclusion

LinkEHR-Ed has been used in several pilot initiatives that have served as a test bed for the technology. These projects have been developed in health delivery institutions, mainly public hospitals, in order to validate not only the technical architecture but also the methodology and whether the real clinical needs are met. The first experience was the normalization of the EHR system of the Hospital General Universitario de Valencia (Spain) according to ISO/CEN EN13606. Five archetypes were defined: biochemical results, surgery procedure, discharge report, infirmary report and patient summary based on the European epSOS [7] specification. Five different departmental databases were used to create archetype instances, one for each archetype. The resulting XQuery transformation scripts were integrated into the EHR engine [8], and executed on-demand to provide with a normalized view of the existing data. A similar experience was developed at the Hospital de Fuenlabrada in Madrid. In this scenario, the primary care EHR and the Spanish Pharmaceutical Specialties Catalog were used to construct a normalized patient summary also based on the epSOS specification. For this purpose, four integration archetypes were defined: medication, problems, alerts and the container patient summary. In all cases the mapping capabilities of LinkEHR-Ed were enough to generate normalized extracts with the intended semantics in a short time. As a performance indicator, we considered the response time for serving a patient summary request. Transformations of source data for a single patient into an ISO/CEN EN13606 XML documents using the generated XQuery script took a fraction of a second, which was found to be insignificant with respect to the data-retrieving time. Finally, a pilot experience between both mentioned hospitals is being carried out; where patient summaries will be interchanged between both hospitals to test the semantic interoperability between both EHR systems.

We have presented LinkEHR-Ed, a tool that allows the utilization of archetypes for upgrading already deployed systems in order to make them compatible with an EHRA standard. The overall objective is to maintain in-production systems and applications without any changes while providing a mean for making public clinical information in the form of standardized EHR extracts, hiding technical details, location and heterogeneity of data repositories. Therefore, archetypes could be used as a semantic layer over the underlying databases associating them with domain specific semantics and therefore upgrading the semantics of the data they hold.

Acknowledgements

This work was supported in part by the Spanish Ministry of Education and Science under Grant TSI2007-66575-C02 and by The Spanish Ministry of Health under Grant RD07/0067/2001.

References

[1] D. Kalra. Electronic health record standards. *Yearbook of medical informatics* (2006), pp. 136-144.
[2] CEN/TC251. EN13606- 2-Health Informatics-Electronic record communication-Part 2 Archetypes.

[3] T. Beale, S. Heard (Ed). Archetype Definition Language ADL 1.4; available from: http://www.openehr.org/releases/1.0.2/architecture/am/adl.pdf
[4] JA. Maldonado, D. Moner, D. Boscá, J.T. Fernández, C. Angulo, M. Robles. LinkEHR-Ed: A multi-reference model archetype editor based on formal semantics, *International Journal of Medical Informatics* **78(8)** (2009), 559-570.
[5] R. Fagin, P.G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering, *Proceedings of the 9th International Conference on Database Theory* (2003), 207-224.
[6] A. Raffio, D. Braga, S. Ceri, P. Papoti, M.A. Hernández. Clip: a Visual Language for Explicit Schema Mappings, 24th Int. Conf. on Data Engineering (2008), 30-39.
[7] European Patients Smart Open Services Project, www.epsos.eu.
[8] C. Angulo, P. Crespo, JA. Maldonado, D. Moner, D. Pérez, I. Abad, J. Mandingorra, M. Robles. Non-invasive lightweight integration engine for building EHR from autonomous distributed systems. *International Journal of Medical Informatics* 2007; **76(S3):**417-424.

Glosario

ADL. Archetype Definition Language

CDA. Clinical Document Architecture

CEN. Comité Europeo de Normalización

DTD. Document Type Definition

EHR . Electronic Health Record

ERP. Enterprise resource planning

EUROREC. European Institute for Health Records

GEF. Graphical Editing Framework

GMM. Graphical Mapping Manager

GUI. Graphical user interface

HL7. Health Level Seven

IDE. Integrated Development Environment

IOM. Institute of Medicine

IopS. Interoperabilidad Semántica

ISO. International Organization for Standardization

MR. Modelo de Referencia

MVC. Model View Controller

PNF. Partitioned Normal Form

RFID. Radio Frequency IDentification

SEMIC. The Semantic Interoperability Centre Europe

SQL. Structured Query Language

TIC. Tecnologías de la Información y la Comunicación

WYSIWYG. What You See Is What You Get

XML. Extensible Markup Language

XSLT. Extensible Stylesheet Language Transformations

Bibliografía

1. **P.Lehman H., A. Abbott P., K.Roderer N., Rothschild A., Mandell S, A.Ferrer J., et al.** Aspects of electronic Health Record Systems. Nueva York : Springer, 2006.
2. **N.Stroetmann V., Kalra D., Lewalle P., Rector A., Rodrigues J.M., A.Stroetmann K., et al.** Semantic Interoperability for Better Health and Safer Healthcare., disponible en: http://ec.europa.eu/information_society/activities/health/docs/publications/2009/2009semantic-health-report.pdf, último acceso septiembre 2009.
3. **Angulo C., Moner D., Maldonado J.A., Reig E., Robles M., Perez D.** Estandarizar la Historia Clínica Electrónica según la norma ISO 13606 con LinkEHR. *Actas del XII Congreso Nacional de Informática de Salud (Infosalud 2009)*. pp. 171-179.
4. **Pressman, Roger S.** Ingeniería del Software. Un enfoque práctico. México D.F. : Mc Graw Hill, 2005.
5. **Beale, Thomas.** Archetypes: Constraint-based Domain Models for Future-proof Information Systems. *Eleventh OOPSLA Workshop on Behavioral Semantics: Serving the Customer*. Boston(2002), pp. 16-32.
6. **Maldonado J.A.** Historia Cínica Federada Basada en la Norma Europea CEN/TC251 EN13606. Tesis doctoral Universidad Politécnica de Valencia. Valencia, (2005).
7. **Open EHR Consortium.** Archetype Definition Language ADL 1.4. Disponible en: http://www.openehr.org/releases/1.0.1/architecture/am/adl.pdf. Último acceso septiembre 2010.
8. **Maldonado J.A. , Moner D., Boscá D., Angulo C., Reig E., Robles M.** Normalización de datos biomédicos basada en arquetipos. *Comunicación del XXV Congreso Anual de la Sociedad Española de Ingeniería Biomédica (CASEIB 2007)*. Cartagena, (2007).
9. **IBIME Group.** LinkEHR-Ed Archetype Editor v0.8 User´s Manual. Disponible en: http://pangea.upv.es/linkehr/wp-content/uploads/2007/07/linkehr-ed-manual.pdf. Último acceso septiembre 2010.
10. **Maldonado J.A., Moner D., Boscá D., Fernández J.T., Angulo C.,Robles M.** LinkEHR-Ed: A multireference model archetype editor based on formal semantics. International Journal of Medical Informatics 78(8)(2009), pp. 559-570.
11. **Raffio A., Braga D., Cesi S., Papotti P., Hernandez M.A.** Clip: a visual Languaje for Explicit Schema Mappings. *International Conference on Data Engineering.* Cancún 7 (12)(2008), pp. 30-39.
12. **Azad B.** Notes on the Eclipse Plug-in Architecture. Disponible en: http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html. Último acceso septiembre 2010.
13. **Eclipse Foundation.** GEF Programmer´s Guide. Disponible en: http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.gef.doc.isv/guide/guide.html. Último acceso septiembre 2010.

14. **Moore B., Dean D., Gerber A., Wagenknecht G., Vanderheyden P.** Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. Disponible en: http://www.redbooks.ibm.com/abstracts/sg246302.html. Último acceso septiembre 2010
15. **Zoio P.** Building a Database Schema Diagram Editor with GEF. Disponible en: http://www.eclipse.org/articles/Article-GEF-editor/gef-schema-editor.html. Último acceso septiembre 2010.
16. **Gamma, Erich.** Patrones de Diseño. Madrid : Addison Wesley, 2006.
17. **Guojie, Jackwind Li.** Java Native Interfaces with SWT/JFace. Danvers MA : Wiley Publishing, Inc., 2005.
18. **Rational, IBM.** Rational Data Architect. Disponible en: http://www.rational.com.ar/herramientas/rationaldataarchitect.html. Último acceso septiembre 2010.
19. **Scruffy-looking Cat Herder**. Disponible en: http://scruffylookingcatherder.com/. Último acceso septiembre 2010.
20. **Weitzenfeld, Alfredo.** Ingeniería del Software Orientada a Objetos con UML, JAVA e Internet. México D.F. : Thomson, 2004.
21. **Sedgewick R.** Algorithms in Java. Stoughton MA : Addison Wesley, 2007.
22. **Seemann J.** Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Towards Automatic Layout of Object-Oriented Software Diagrams. Lecture Notes in Computer Science volume 1353/1997, pp. 415-424.
23. **Maldonado J.A., Moner D., Marco L.,Fernández-Breis J.T., Reig E. et al.** LinkEHR: A Tool for Standardization and Integration of Legacy Clinical Data. EFMI Special Topic Conference 2010. Reykjavik, Islandia.