Recuperación, transformación y simulación de datos biológicos mediante ingeniería dirigida por modelos.

Abel Gómez Llana

Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación
Cno. de Vera, s/n. 46022 Valencia.

Dirigido por

José Á. Carsí
Isidro Ramos

b

Índice general

Índice de figuras

Listados de código

Capítulo 1

Introducción

La práctica tradicional en los estudios científicos de «experimentación → análisis → publicación» está cambiando a «experimentación → organización de datos → análisis → publicación». Esto se debe a que, hoy en día, los datos no se obtienen únicamente de experimentos, sino que también se obtienen de simulaciones. Esta gran cantidad de datos generados no siempre tiene la misma estructura y puede estar almacecenada en bases de datos heterogéneas. Además, el elevado número de datos requiere el desarrollo de herramientas informáticas que nos permitan representarlos para analizarlos, y a su vez, realizar otras simulaciones con ellos.

En el campo de la bioinformática encontramos esta problemática en el análisis y simulación de los mecanismos de señales de las células (*Pathways*). Un *pathway* es un conjunto de reacciones químicas que se producen en el interior de una célula tras la recepción de un estímulo. En estudios de este tipo se observa tanto el rellenado independiente de fuentes de datos como el desarrollo independiente de herramientas de modelado. Por ello, estos datos deben ser reconvertidos de forma manual para que puedan ser utilizados en las herramientas de simulación. En un escenario así, disponer de herramientas interoperables es deseable.

El Desarrollo de Software Dirigido por Modelos —DSDM— es una aproximación que pretende dar solución a problemas como éste. En el DSDM, un modelo es una estructura de datos que puede ser definida mediante un lenguaje de modelado (generalmente llamado metamodelo). Un modelo permite definir la funcionalidad, estructura y/o comportamiento de un sistema [47] dependiendo del metamodelo utilizado. La utilización de modelos en un proceso de DSDM permite automatizar el desarrollo de aplicaciones y su evolución mediante técnicas de programación generativa [9], como las transformaciones de modelos y la generación de código.

En el seno del grupo de investigación ISSI [68] se ha trabajado durante los últimos años en el desarrollo de un *framework* de gestión de modelos que emplea como *back-end* un sistema de reescritura de términos. Este *framework* se ha sido materializado en una herramienta llamada MOMENT (descrita en mayor detalle en la sección 4.2). Esta

herramienta se ha construído sobre Eclipse [51] y EMF [18] dando soporte a esta nueva aproximación en el proceso de desarrollo de software.

1.1. Objetivos

Este trabajo pretende demostrar cómo la filosofía de DSDM permite resolver los problemas surgidos en el estudio de los caminos de señales en el campo de la bioinformática. Los problemas de interoperabilidad entre aplicaciones pueden abordarse de una manera sistemática, donde:

- El formato de datos puede definirse como un modelo
- Los datos se definen como una colección de objectos que son instancia de las clases de este modelo.
- Los datos pueden tratarse desde la perspectiva del DSDM (por ejemplo, usando transformaciones de modelos para tratar con ellos.

De esta manera, podremos alcanzar los siguientes objetivos:

- Mejorar la calidad del software producido, obteniendo herramientas más modulares.
- Automatizar el proceso de migración de datos.
- Independizar el formato de persistencia de los mecanismos de transformación aplicados sobre ellos.
- Elevar el nivel de abstracción en el proceso de transformación, facilitando la colaboración y el intercambio de información entre biólogos y desarrolladores de software.
- Proporcionar capacidades de trazabilidad.
- Finalmente, y como resultado de los factores anteriores, reducir el coste de producción de las herramientas afectando en la productividad de los usuarios biólogos.

Finalmente, se emplearán diversas herramientas para dar soporte a esta aproximación. En primer lugar, se empleará MOMENT (y su soporte al lenguaje QVT) para depurar la herramienta desde el punto de vista del usuario como una aproximación inicial. Con ello se persiguen los siguientes objetivos concretos, que deben reflejarse en un documento escrito:

- Comenzar a implementar la aproximación, dada la pronta aparición de este prototipo (a los pocos meses de la aparición del estándar QVT). Estudiar la viabilidad de la solución aportada.

- Especificar el proceso de instalación y puesta a punto en funcionamiento de la herramienta.
- Identificar las restricciones y convenciones de uso que deben ser respetadas por los usuarios para el correcto funcionamiento de la herramienta. En esto se engloban las restricciones y guías que deben seguirse en la definición de los metamodelos en EMF, así como en la definición y convenciones de nombrado en la especificación de transformaciones.
- Describir el proceso de ejecución de transformaciones para ayudar a los usuarios en el proceso de depuración.
- Documentar la interfaz de usuario, especificando claramente y paso a paso cómo se deben ejecutar las transformaciones desde el punto de vista del usuario final.

Por otra parte, se empleará el motor de transformaciones MediniQVT, como una implementación de terceras partes que da soporte al lenguaje QVT–Relations. En este sentido, como únicamente el motor es libre (y no sus herramientas asociadas) será necesario desarrollar un conjunto de plugins para integrar este motor y producir un prototipo final. Los objetivos concretos en este caso serán desarrollar un prototipo completo, funcional, y completamente automatizado que implemente por completo la propuesta realizada.

1.2. Descripción del documento

El documento se organiza de la siguiente manera. En primer lugar, el capítulo 2 describe el estado del arte en el estudio de los llamados *mecanismos de señales*. En segundo lugar, el capítulo 3 presenta el estado del arte actual en Ingeniería Dirigida por Modelos. En tercer lugar, el capítulo 4 describe cómo se ofrece soporte tecnológico a todas las ideas presentadas en el capítulo 3. El siguiente capítulo, el capítulo 5 describe el caso de estudio concreto, así como la solución propuesta mediante Ingeniería Dirigida por Modelos para abordarlo. En el capítulo 6 se describe en profundidad como se ha llevado a cabo la implementación de la propuesta. En este sentido se describen en detalle los modelos, metamodelos, proceso de pre- y postprocesado, reglas de transformación definidas para las diferentes herramientas, etc. Los siguientes capítulos, los capítulos 7 y 8 describen en profundidad cómo se han de emplear las diferentes herramientas desarrolladas. En este sentido, el capítulo 7 describe cuáles son las restricciones y convenciones que se han detectado que deben seguir al emplear la herramienta MOMENT–QVT, y el capítulo 8 describe cómo se debe usar el prototipo implementado utilizando la biblioteca MediniQVT. Las secciones 9 y 10 muestras las conclusiones finales y agradecimientos a terceras personas respectivamente. Finalmente, se presentan diversas informaciones adicionales (instancias de ejemplo, código completo de las transformaciones y fragmentos de código significativos) en los Anexos A, B, C, D, E, F, G, H, I y J.

Capítulo 2

Estudio de los mecanismos de señales.

2.1. Motivación.

Numerosos procesos biológicos son controlados por los denominados *mecanismos de señales*. Un camino de señales —*signaling pathway*— es una colección de reacciones químicas que tienen lugar en una célula como reacción a un estímulo (generalmente externo) disparando diversos eventos en el interior de ésta. Estos eventos son los que controlan la progresión del ciclo de vida (muerte, crecimiento, especialización), activación o desactivación de genes para el momento de la reproducción, el movimiento de la célula, etc.

Un estímulo puede ser, por ejemplo, la presencia de una determinada molécula en el entorno de la célula. Dependiendo de la naturaleza de esta célula, el estímulo puede desencadenar una respuesta del sistema inmunitario. El conocimiento de cómo el sistema inmunitario responde a un ataque externo es muy interesante, por ejemplo, en el campo de la medicina. Conocer los mecanismos infecciosos ayuda en el desarrollo de nuevo fármacos. De esta manera, se pueden formular remedios mas eficaces y específicos para determinados antígenos. Igualmente, esto reduciría los riesgos en los estudios experimentales con nuevos medicamentos. Actualmente hay ciertos comportamientos del sistema inmunitario humano que no se encuentran en otros animales, imposibilitando los experimentos clínicos con ellos antes de aplicarse a seres humanos. En este caso el estudio teórico de los *pathways* es interesante ya que previa simulación, sería posible prever el efecto producido sobre el organismo de las sustancias que formen parte de una determinada formulación.

2.2. Aproximaciones actuales

El estudio de estos caminos de señales se realiza mediante el ensayo empírico de los estímulos externos que recibe una célula y recogiendo los datos sobre las reacciones químicas

desencadenadas en su interior. Estos datos obtenidos de forma experimental por los biólogos son almacenados en diferentes bases de datos para su posterior estudio y consulta por otros expertos. Ejemplo de ello son bases de datos como Transpath®, BioCarta, KEGG o Reactome. A pesar de que éstas son algunas de las más representativas, cada vez existen un numero creciente de ellas. Según [25] existen actualmente 968 bases de datos sobre datos moleculares y biológicos, 110 más que en 2006, y 249 más que en 2005.

Para el estudio, modelado y análisis de los procesos biológicos almacenados en estas bases de datos se han adaptado y aplicado algunos de los formalismos desarrollados para el análisis de sistemas concurrentes. Eso es debido a que las reacciones químicas involucradas en un *pathway* suelen ser un proceso en cascada, produciéndose comunmente reacciones de forma concurrente.

En π-cálculo, por ejemplo, los modelos representan las proteínas como procesos activos que reaccionan cuando son combinados con los procesos adecuados [59]. BioSPI es una herramienta que implementa una variante estocástica de π-cálculo y ha sido usada para simular algunos aspectos probabilísticos y cinéticos de reacciones bio-químicas [55]. BioAmbients [58] es una adaptación de ambient calculus para el modelado de la dinámica de compartimientos biológicos. Modelos BioAmbients pueden ser simulados haciendo uso de una extensión de la herramienta BioSPI. Técnicas de análisis de programas se han utilizado para analizar modelos BioAmbients [46], por ejemplo para demostrar que de acuerdo a un modelo dado una proteína nunca aparecerá en un compartimiento. Diagrama de estado expresan de forma natural la división en compartimientos y los procesos jerárquicos así como el control del flujo entre subprocesos [35, 14]. Life Sequence Charts (LSCs) es un formalismo visual que permite especificar sistemas reactivos en término de su comportamiento inter-objetual [10]. Está soportado por una herramienta llamada Play-Engine [28] que permite construir los modelos de forma gráfica. El Play-Engine también soporta la animación y el análisis de modelos LSC. LSCs y Play-Engine ya han sido aplicados de forma exitosa al modelado de sistemas biológicos [36]. Las redes de Petri han sido extensamente utilizadas en sistemas biológicos [75, 30]. Las redes de Petri coloreadas (CPNs) permiten representar la estructura, proporcionando una forma natural para modelar modificaciones post-translacionales de proteínas y complejos. Las herramientas para trabajar con CPN soportan la construcción gráfica de CPNs así como su simulación. La Pathway Logic (PL [2]) [16, 17, 63] es una aproximación al modelado de procesos celulares y moleculares basado en la lógica de reescritura [45]. La estructura de las células y sus componentes son representados mediante tipos de datos y los procesos celulares mediante reglas de reescritura en el lenguaje Maude [70]. El asistente de Pathway Logic proporciona una representación visual interactiva de los modelos PL que permite a los biólogos navegar y hacer consultas a la base de conocimiento PL en el contexto de información biológica.

En la actualidad también se están desarrollando un número creciente de notaciones gráficas para las interacciones y reacciones que se producen. Cell Designer [38] y Molecular Interaction Maps proporcionan notaciones visuales semi-formales para representar redes de interacciones moleculares. En la actualidad no parece haber una representación computacional asociada a la interacción de las redes. En [11] se presenta una notación gráfica con

su semántica formal asociada en término de cálculo de procesos para reacciones básicas. Cardelli, en [8], presenta diferentes notaciones gráficas para diferentes tipos de procesos tales como las reacciones de transporte en la membrana, cinética química y la regulación de genes. Según nuestro conocimiento, hasta la fecha, no existen herramientas disponibles para usar dichas notaciones en la descripción de modelos. Además de las notaciones para describir reacciones individuales, también son necesarios lenguajes para describir las propiedades de los caminos de señales y así poder ensamblar y consultar los modelos. Las notaciones existentes incluyen lenguajes de consultas para bases de datos y lógicas formales temporales. Todas inapropiadas para ser usadas por biólogos.

Como se puede observar, el estado actual del arte muestra un escenario donde las fuentes de datos se rellenan de forma independiente, teniendo formatos dispares y sólo capturando cierta información relevante. Igualmente, las herramientas de simulación, por ejemplo para el modelado de señales, también han sido desarrolladas de forma aislada y requiriendo que los datos de entrada se proporcionen en un formato específico y no estándar. Esto provoca que, como un famoso biólogo expresó hace poco «El estado actual del arte está convirtiendo a los biólogos en programadores Perl» [19].

La situación deseable sería proporcionar a los biólogos herramientas específicas de dominio, con lenguajes sencillos, declarativos y cercanos a su saber hacer tradicional que les permita realizar el trabajo de forma simple. Igualmente, se debe buscar apoyo en los estándares desarrollados para garantizar que el trabajo realizado, y los resultados obtenidos mediante estas herramientas esté en un formato que no requiera un fuerte proceso de reconversión para que pueda ser reutilizado.

2.3. Caso de estudio.

El caso de estudio parte de los trabajos realizados por Taübner et al. [64], donde se ha modelado, simulado y validado el *pathway* del *receptor de tipo toll 4* (TLR4) —que se introduce en el siguiente punto— mediante redes de Petri coloreadas. A continuación se expone en primer lugar el contexto biológico, explicando los receptores de tipo *Toll* y más específicamente el receptor de tipo *Toll* 4. En segundo lugar se describen los detalles tecnológicos del estudio de éste realizados en [64]: fuente de datos y herramienta de simulación seleccionadas y proceso de extracción y tratamiento de los datos.

2.3.1. Los receptores de tipo *Toll*.

Los receptores Toll fueron identificados por vez primera como una molécula esencial que determina el eje dorso-ventral en el desarrollo embrionario de Drosophila[1] [29]). Pero se observó que Toll no sólo tenía funciones de desarrollo embrionario, ya que también poseía funciones inmunes, al comprobar que las moscas que no lo poseían, morían por infecciones

[1] Comunmente llamada «mosca del vinagre».

fúngicas. Su nombre lo acuñó Nusslein-Volhard a partir de la palabra alemana Toll, que significa «extraño o fantástico» [3], ya que las moscas que no lo tenían funcional, presentaban un desarrollo anormal. En 1997 se identificó el primer homólogo humano del receptor Toll de Drosophila y se denominó receptor Toll-like (TLR) [42]. Se trata una familia de proteínas trasmembrana de tipo I que forman parte del sistema inmunitario. En los vertebrados también posibilitan la adaptación del sistema inmune. Descubiertos inicialmente de la mosca de la fruta (Drosophila melanogaster) son los responsables del reconocimiento de varias vías de patrones de reconocimieno de patógenos (PAMPs — pathogen-associated molecular patterns) expresados por un amplio espectro de agentes infecciosos. Monocitos/macrófagos y neutrófilos fagocitan patógenos microbianos y estimulan la respuesta de citocinas[2] dando como resultado el desarrollo de la inmunidad innata o natural, la respuesta inflamatoria y median la efectiva inmunidad adaptativa. Su función, en resumen, es el reconocimiento del patógeno y la estimulación de la respuesta inmunitaria contra dichos patógenos.

También han sido encontrado en plantas, teniendo un origen evolutivo muy antiguo; después de las defensinas[3], pueden ser el componente del sistema inmune más antiguo.

Se ha estimado que la mayoría de las especies de mamíferos tienen entre diez y quince tipos de receptores de tipo *Toll*. Actualmente, se han identificado trece TLRs (denominados simplemente TLR1 a TLR13) tanto en humanos como en ratones, y formas equivalentes de estos se han encontrado en otras especies de mamíferos. No obstante, no se han encontrado equivalencias en todos los mamíferos para ciertos TLR encontrados en el ser humano[42, 12, 62]. Por ejemplo, una proteína para la codificación de genes análoga al TLR10 de los humanos se encuentra en los ratones, pero parece que en un pasado se dañó devido a un retrovirus. Por otra parte, los ratones expresan los TLRs 11, 12 y 13, ninguno de los cuales se representa en los humanos. Otros mamíferos pueden expresar TLRs que no se encuentran en humanos. Estos puede complicar el uso de animales como modelos para experimentar con el sistema inmunitario humano.

Dado que la especificidad los receptos de tipo *Toll* (y otros receptores del sistema inmune innato) no puede cambiarse fácilmente en el curso de la evolución, estos receptores reconocen moléculas que están constantemente asociadas a ataques contra el organismo (por ejemplo, producidos por patógenos o por estrés celular) y son altamente específicos de estos ataques (por lo que no pueden confundirse con las células propias del organismo). Las moléculas asociadas a patógenos que cumplen este requerimiento son generalmente críticas para las funciones de éste, y por tanto, no pueden eliminarse o cambiar a causa de una mutación. Estas moléculas están obligadas a mantenerse a lo largo de la evolución. Estas caraterísticas bien conservadas en los patógenos, incluyen lipopolisacáridos (LPS), lipoproteinas, lipopéptidos y lipoarabinomannan que se encuentran en la superficie de las células bacterianas; proteínas como la flagelina que se encuentra en los flagelos de bacterias;

[2]Las citocinas son proteínas que regulan la función de las células que las producen u otros tipos celulares, sinedo los agentes responsables de la comunicación intercelular.

[3]Las defensinas son pequeñas proteínas ricas en iones que se encuentran en vertebrados e invertebrados y funcionan como antibióticos naturales que se hayan en la superficie de la piel. Son activas contra bacterias, hongos y virus enclaustrados.

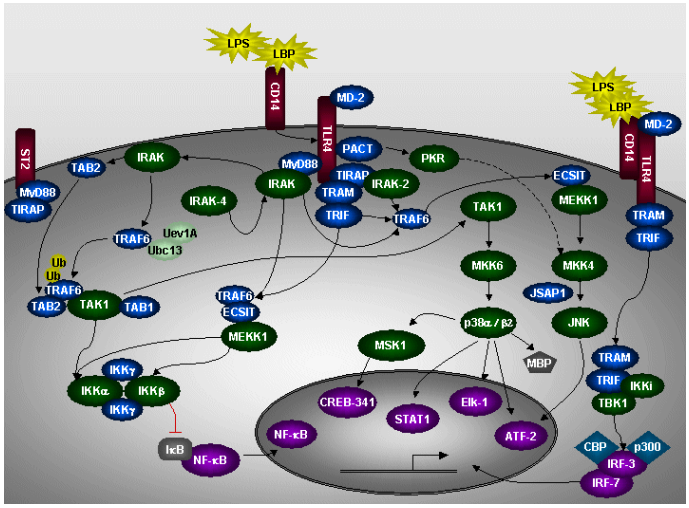


Figura 2.1: Representación gráfica del pathway TLR4.

dobles cadenas de ARN de los virus o las regiones no metiladas (llamadas «islas CpG») de ADN viral o bacteriano; etc.

2.3.1.1. El *pathway* del receptor de tipo *Toll* 4.

El receptor de tipo *Toll* 4 fue el primero en ser estudiado y es el más conocido [42]. Participa en el reconocimiento de lipopolisacáridos (LPS), componente esencial de las bacterias Gram-negativas. Para emitir las señales necesarias para iniciar una respuesta inmune requiere otras moléculas adicionales (ver figura 2.1).El LPS se une a una proteína de unión a LPS presente en el suero (LBP). El complejo es luego reconocido por el receptor CD14, una molécula anclada por glucosilfosfatidilinositol que se expresa en forma preferencial en los monocitos, macrófagos y neutrófilos. La estimulación por LPS va seguida de acercamiento físico del CD14 y TLR4. La MD-2, una molécula que se asocia con la porción extracelular del TLR4, aumenta la respuesta al LPS. La participación de la proteína quedó confirmada al observarse una menor respuesta a LPS en ratones con deficiencia de MD-2, cuyos macrófagos, células dendríticas y linfocitos B casi no responden al LPS. Más aun, los ratones con deficiencia son resistentes a la inducción de sepsis por LPS y se ha visto que la proteína es crucial para la distribución intracelular del TLR4. Otra proteína de superficie —RP105— también está involucrada en el reconocimiento de LPS. La RP105 contiene un dominio rico en lucina estructuralmente relacionado con la porción extracelular de los TLR y se expresa fundamentalmente en linfocitos B. La proteína de fusión del virus respiratorio sincicial es otro de los ligandos del TLR4 y CD14. Además se ha visto que el TLR4 reconoce algunos ligandos endógenos. Los autores recuerdan que las proteínas de shock térmico tienen una estructura muy conservada desde bacterias hasta mamíferos. Varios estresantes, como shock térmico, radiación ultravioleta e infección por bacterias y virus aumentan la síntesis de proteínas de shock térmico. Estas proteínas activan macrófagos y células den-

dríticas a secretar citoquinas proinflamatorias y a expresar moléculas coestimulatorias. Por este motivo se considera que las proteínas de shock térmico podrían representar una señal endógena de peligro.

La función de la proteína de shock térmico (HSP60) es la que mejor se conoce. Su acción está mediada por el TLR4 y ha sido involucrada en la inflamación que acompaña la arteriosclerosis, cuya aparición se relacionó con la infección por *Chlamydia pneumoniae* (una bacteria que se transmite al estornudar o al toser). Más aun, se considera que esta HPS60 es uno de los factores que promueven enfermedad vascular en asociación con la infección. La HSP60 inducida por clamidias activa células de músculo liso vascular y macrófagos a través del TLR4. Sin embargo, la evidencia en conjunto sugiere que el TLR4 no está directamente involucrado en el reconocimiento de la HPS60.

En respuesta al daño tisular (daño de los tejidos del organismo) se producen diversos componentes de la matriz extracelular -fibronectina, ácido hialurónico y heparán sulfato- que parecen importantes en el proceso de remodelamiento tisular. El dominio A de la fibronectina tiene actividades inmunoestimulantes similares a las ocasionadas por LPS; en la respuesta a dicho dominio está involucrado el TLR4. Asimismo, la activación de células dendríticas por ácido hialurónico está mediada por TLR4.

Los depósitos de fibrina extravascular (derivados del fibrinógeno plasmático) son marcadores de inflamación en el contexto de daño, infección y trastornos inmunológicos. La capacidad del fibrinógeno de inducir producción de quemoquinas por los macrófagos está mediada por el TLR4.

Los LPS también son reconocidos independientemente del TLR4. Algunas de las moléculas que participan en esta unión son la HSP70, la HSP90 y el receptor de quemoquina 4 (CXCR4). Una vez que el LPS se une a la superficie de la célula se libera rápidamente en el citoplasma y este movimiento intracelular parece esencial para algunas funciones. Los estudios indican que el LPS puede ser reconocido tanto en la superficie celular como en el interior de las células, probablemente a través de la proteína Nod1. Esta proteína activa el NF-KAPPAB en respuesta a LPS, lo cual sugiere que el Nod1 es un receptor para LPS en el interior de las células.

Para el caso de estudio, por simplicidad, se toma como ejemplo una pequeña parte del *pathway* en el que interviene el receptor de tipo *Toll* 4 —TLR4—, por ser el primero reconocido en mamíferos y el mejor estudiado hasta el momento. A continuación se indican las reacciones que serán transformadas en el caso de estudio.

1. LPS + LBP $\leftrightharpoons$ LPS:LBP

2. LPS:LBP + CD14 $\leftrightharpoons$ LPS:LBP:CD14

3. ST2 + TIRAP $\leftrightharpoons$ ST2:TIRAP

4. ST2 + MyD88 $\leftrightharpoons$ ST2:MyD88

La notación expresa a la izquierda los reactivos y a la derecha los productos. El símbolo «⇋» expresa que esta reacción es bidireccional, ya que se trata de una reacción de *equilibrio químico* —informalmente se puede describir como una reacción que se mantiene hasta que los reactivos y los productos alcanzan un estado de equilibrio, no llegándose a consumir ninguno de los dos—. Para `(1)`, por ejemplo, podríamos expresar de forma sencilla lo siguiente: *la mólecula LPS se une a la molécula LBP, dando lugar a la molécula compuesta LPS:LBP.*

2.3.2. Aproximación actual en el estudio del pathway del TLR4 en bioinformática.

Son numerosas las fuentes de datos de las que se puede extraer información sobre el *pathway* del *TLR4*. Algunas de ellas pueden ser TRANSPATH®, KEGG, Reactome o BioCarta. Estas bases de datos han sido rellenadas a partir de estudios empíricos y pueden contener información incompleta e incluso contradictoria. Los actuales trabajos para la representación y simulación de pathways se dirigen en la representación y simulación de estos *pathways* para poder validar la información obtenida en los estudios empíricos.

Con ese objetivo se han adaptado diversos formalismos gráficos para la simulación de *pathways* biológicos. Ejemplos de ello son los diagramas de estados [65], los *Life Sequence Charts* [65, 10], o las redes de Petri coloreadas [64]. El uso de redes de Petri para el análisis de procesos biológicos permite representar las siguientes vistas según Peleg, Zeh y Altman [54]:

- Vista estática (estructural): Moléculas que participan en el sistema, propiedades y relaciones entre ellas.
- Vista dinámica: procesos secuenciales, paralelos, condicionales e iterativos.
- Vista funcional: muestra los actores, por ejemplo, enzimas que realizan cada función, sustratos y productos de cada función, etc.

El trabajo en el que se fundamenta la solución aquí presentada describe un proceso mediante el cual es posible extraer los datos del pathway TLR4 de la base de datos TRANSPATH®con el objetivo de simularlo mediante la herramienta de simulación y validación de redes de Petri coloreadas *CPN Tools*.

2.3.2.1. La base de datos TRANSPATH®.

TRANSPATH®es una base de datos comercial desarrollada y mantenida por la empresa alemana BIOBASE[1]. TRANSPATH®almacena información acerca de caminos de señales y en su versión 7.4 contiene información sobre 62.549 moléculas, 23.076 genes y

TRANSPATH MOLECULE TABLE, Release 6.0

Statistics: Number of complexes 2; Number of reactions 3

Jump to Visualization

Accession number: **MO000022132**

Created 2001-11-05 by: Claudia Choi (click for feedback).

Updated 2002-12-03 by: Mathias Krull (click for feedback).

Copyright: Copyright (c) Biobase GmbH.

Molecule name: **TLR4(m)**

Synonyms: TLR-4; toll-like receptor 4.

Encoding gene: G006820; TLR4.

Species: mouse, Mus musculus.

Classification: membrane-transducing components; receptors; pathogen-recognition receptors; TLRs; TLR4 .
membrane-transducing components; receptors; cytokine receptor family; TIR superfamily; TLRs; TLR4.

Type: basic.

Superfamilies: MO000019394; TLR4.

Sequence length, molecular weight: 835 AA; 95.5 kDa (cDNA)(calc.).

Isoelectric point: 6.09 (calc.).

Sequence source: translated from EMBL:AF185285.

Sequence:

```
MMPFWLLART LIMALFFSCL TPGSLNPCIE VVPNITYQCM DQKLSKVPDD IPSSTKNIDL
SFNPLKILKS YSFSNFSELQ WLDLSRCEIE TIEDKAWHGL HHLSNLILTG NPIQSFSPGS
FSGLTSLENL VAVETKLASL ESFPIGQLIT LKKLNVAHNF IHSCKLPAYF SNLTNLVHVD
LSYNYIQTIT VNDLQFLREN PQVNLSLDIS LNPIDFIQDQ AFQGIKLHEL TLRGNFNSSN
IMKTCLQNLA GLHIHRLILG EFKDERNLEI FEPSIMEGLC DVTIDEFRLT YTNDFSDDIV
KFHCLANVSA MSLAGVSIKY LEDVPKHFKW QSLSIIRCQL KQFPTLDLPF LKSLTLTMNK
GSISFKKVAL PSLSYLDLSR NALSFSGCCS YSDLGTNSLR HLDLSFNGAI IMSANFMGLE
ELQHLDFQHS TLKRVTEFSA FLSLEKLLYL DISYTNTKID FDGIFLGLTS LNTLKMAGNS
FKDNTLSNVF ANTTNLTFLD LSKCQLEQIS WGVFDTLHRL QLLNMSHNNL LFLDSSHYNQ
LYSLSTLDCS FNRIETSKGI LQHFPKSLAF FNLTNNSVAC ICEHQKFLQW VKDQKQFLVN
VEQMTCATPV EMNTSLVLDF NNSTCYMYKT IISVSVVSVI VVSTVAFLIY HFYFHLILIA
GCKKYSRGES IYDAFVIYSS QNEDWVRNEL VKNLEEGVPR FHLCLHYRDF IPGVAIAANI
IQEGFHKSRK VIVVVSRHFI QSRWCIFEYE IAQTWQFLSS HSGIIFIVLE KVEKSLLRQQ
VELYRLLSRN TYLEWEDNPL GRHIFWRRLK NALLDGKASN PEQTAEEEQE TATWI
```

External database hyperlinks:
MGD: MGI:96824; .
SWISSPROT: Q9QZF5; .
SWISSPROT: Q9QUK6; .
BKL: MousePD:Tlr4.
INTERPRO: IPR000157; TIR. [3].
INTERPRO: IPR000887; KDPG and KHG aldolase. [3].
INTERPRO: IPR000483; Cysteine-rich flanking region, C-terminal. [3].
INTERPRO: IPR001611; Leucine-rich repeat. [3].

External database hyperlinks (of encoding gene): show

Features (motifs):

54	77	PFAM: PF00560; Leucine Rich Repeat; 9.5; 0.46; [4].
78	101	PFAM: PF00560; Leucine Rich Repeat; 20.8; 0.00023; [4].
102	125	PFAM: PF00560; Leucine Rich Repeat; 20.6; 0.00026; [4].
150	174	PFAM: PF00560; Leucine Rich Repeat; 12.2; 0.086; [4].
175	198	PFAM: PF00560; Leucine Rich Repeat; 16.9; 0.0035; [4].
398	420	PFAM: PF00560; Leucine Rich Repeat; 9.4; 0.47; [4].
495	518	PFAM: PF00560; Leucine Rich Repeat; 19.3; 0.00064; [4].
576	626	PFAM: PF01463; Leucine rich repeat C-terminal domain; 24.8; 1.4e-05; [4].
674	812	PFAM: PF01582; TIR domain; 149.5; 4.2e-43; [4].

Scheme of

GO: biological process, molecular function: show

Location positive and experiment(s) More info...:
GO: cellular_component: extracellular space; TAS; GO:0005615.
GO: cellular_component: lipopolysaccharide receptor complex; ISS; GO:0046696.

Complexes: show

Reaction:
XN000005346; TLR4(m) + MD-2(m) <=> TLR4(m):MD-2(m) (binding) [1].
XN000005380; TLR4(m) + MyD88(m) <=> TLR4(m):MyD88(m) (binding) [2].

Reaction upstream: XN000020629; TLR4(m) -> TLR4(m) (expression).

Visualization: Show graphics for the signaling network — In this window | In a new window

Reference number Author(s), Title, Journal:

[1] PUBMED link and title in TRANSPATH Professional only..
Akashi S., Shimazu R., Ogata H., Nagai Y., Takeda K., Kimoto M., Miyake K.
J. Immunol. 164:3471-3475 (2000).

[2] PUBMED link and title in TRANSPATH Professional only..
Rhee S. H., Hwang D.
J. Biol. Chem. 275:34035-34040 (2000).

[3]
Mulder N.J. et al.
Nucl. Acids. Res. 31: 315-318.

[4]
Philip Stegmaier, TRANSPATH_Team.
TRANSPATH Reports 1:0001 (2003).

Figura 2.2: Información de la molécula TLR4 mostrada en la interfaz web de TRANSPATH®.

```
//
AC  MO0000022132
XX
DT  2001-11-05 15:38:40.0(created); cch.
DT  2002-12-03 09:39:50.0(updated); mkl.
CO  Copyright (c) Biobase GmbH.
XX
NA  TLR4(m)
SY  TLR-4; toll-like receptor 4.
GE  <G006820>; TLR4.
OS  mouse, Mus musculus.
XX
CL  membrane-transducing components; receptors; cytokine receptor family; TIR superfamily; TLRs; TLR4
tion receptors; TLRs; TLR4.
TY  basic.
HP  <MO0000019394>; TLR4.
XX
SZ  835 AA; 95.5 kDa (cDNA) (calc.).
IP  6.09 (calc.).
SC  translated from EMBL:AF185285.
SQ  MMPPWLLART LIMALFFSCL TPGSLNPCIE VVPNITYQCM DQKLSKVPDD
SQ  IPSSTKNIDL SFNPLKILKS YSFSNFSELQ WLDLSRCEIE TIEDKAWHGL
...
SQ  GRHIFWRRLK NALLDGKASN PEQTAEEEQE TATWT
XX
DR  <MGD:MGI:96824>; .
DR  <SWISSPROT:Q9QZF5>; .
DR  <SWISSPROT:Q9QUK6>; .
DR  <BKL:HumanPSD:Tlr4; Mouse>.
DR  <InterPro:IPR000157>; TIR. [3].
DR  <InterPro:IPR000887>; KDPG and KHG aldolase. [3].
DR  <InterPro:IPR000483>; Cysteine-rich flanking region, C-terminal. [3].
DR  <InterPro:IPR001611>; Leucine-rich repeat. [3].
XX
DR  {GENOMIC}AFFYMETRIX:AFFY_MG_U74:38049.
...
DR  {GENOMIC}<UNIGENE:Mm.85343>.
XX
GO  GO: biological_process: I-kappaB kinase/NF-kappaB cascade; GO:0007249.
...
GO  GO: molecular_function: catalytic activity; GO:0003824.
...
XX
CP  GO: cellular_component: extracellular space; GO:0005615.
CP  GO: cellular_component: integral to membrane; GO:0016021.
CP  GO: cellular_component: lipopolysaccharide receptor complex; ISS; GO:0046696.
CP  GO: cellular_component: membrane; IEA; GO:0016020.
CX  <MO0000022133>; TLR4(m):MD-2(m).
CX  <MO0000022180>; TLR4(m):MyD88(m).
XX
XA  <XN0000005346>; TLR4(m) + MD-2(m) <=> TLR4(m):MD-2(m) (binding) [1].
XA  <XN0000005380>; TLR4(m) + MyD88(m) <=> TLR4(m):MyD88(m) (binding) [2].
XB  <XN0000020629>; TLR4(m) -> TLR4(m) (expression).
XX
RN  [1].
RX  <pubmed:10725698>.
RA  Akashi S., Shimazu R., Ogata H., Nagai Y., Takeda K., Kimoto M., Miyake K.
RT  Cutting edge: cell surface expression and lipopolysaccharide signaling via the toll-like receptor
RL  J. Immunol. 164:3471-3475 (2000).
```

Figura 2.3: Información de la molécula TLR4 mostrada como un fichero de texto plano de TRANSPATH®.

104.362 reacciones. Permite extraer la información de diversas maneras: mediante consultas a la base de datos a través de una interfaz web (figura 2.2), peristiendo la información directamente de la base de datos sobre archivos de texto plano2.3 o documentos XML2.4.

En [64] se emplea la representación en XML para extraer la información del *pathway* del TLR4 de la base de datos TRANSPATH®. Esta será también la representación que emplearemos en nuestro caso de estudio. En el anexo D se encuentra la información completa del *pathway* TLR4 en su representación en XML.

2.3.2.2. Redes de Petri coloreadas.

Una red de Petri, por su parte, es una representación formal para un sistema distribuido discreto, que permite representar eventos concurrentes. La red se forma por nodos (llamados lugares), transiciones y arcos dirigidos. Estos arcos conectan siempre un lugar con una transición o una transición con un lugar. En los lugares puede haber un determinado número de tokens, que pueden moverse de un lugar a otro cuando una transición se dispara (una transición se habilita cuando todas sus entradas contienen tokens). La figura 2.5 muestra un ejemplo de red de Petri, donde los círculos blancos son los lugares, el rectángulo relleno de color negro es una transición, las flechas son los arcos dirigidos, y los puntos negros, los tokens. Como se observa en la figura, las redes de Petri coloreadas son muy adecuadas para la simulación de reacciones químicas como procesos unidireccionales, ya que permiten simular de forma muy simple cómo los átomos y moléculas se combinan formando nuevos compuestos.

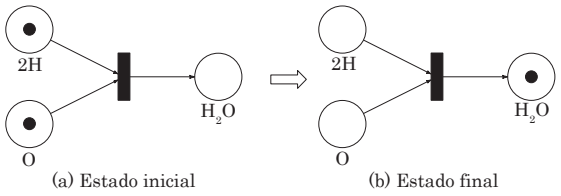


Figura 2.5: Ejemplo de aplicación de redes de Petri en la simulación de reacciones químicas.

Una red de Petri coloreada es aquella en la que los tokens pueden tener un determinado color (esto es, algún atributo distintivo), pudiendo caracterizar tokens de diferentes tipos. *CPN Tools* es una herramienta que permite la construcción, modificación y validación sintáctica de redes de Petri coloreadas de forma gráfica. Dispone además de un simulador (tanto interactivo como automático) para poder inspeccionar la evolución de estos modelos.

La figura 2.6 muestra una captura de pantalla de la herramienta *CPN Tools* mostrando la representación de las cuatro reacciones del caso de estudio.

```
<Molecule id="MO0000022132">
<!-- Copyright (c) Biobase GmbH -->
 <creator>cch</creator>
 <updator>mkl</updator>
 <type>basic</type>
 <name>TLR4(m)</name>
 <synonyms>TLR-4</synonyms>
 <synonyms>toll-like receptor 4</synonyms>
 <firstpos>0</firstpos>

<lastpos>0</lastpos>
 <klass>membrane-transducing components; receptors; pathogen-recognition receptors; TLRs;
receptor family; TIR superfamily; TLRs; TLR4</klass>
 <seq_source>translated from EMBL:AF185285</seq_source>
 <seq_len>835</seq_len>
 <seq_mw>95.5 kDa (cDNA)(calc.)</seq_mw>
 <seq_ip>6.09</seq_ip>
 <sequence>MMPPWLLARTLIMALFFSCLTPGSLNPCIEVVPNITYQCMDQKLSKVPDDIPSSTKNIDLSFNPLKILKSYSFSNFSE
LESFPIGQLITLKKLNVAHNF ....
 <species>mouse, Mus musculus;
 </species>
 <accnos>MGD:MGI:96824; ()</accnos>
 <accnos>SWISSPROT:Q9QZF5; ()</accnos>
 <accnos>SWISSPROT:Q9QUK6; ()</accnos>
 <accnos>BKL:HumanPSD:Tlr4; Mouse</accnos>
 <accnos>InterPro:IPR000157; TIR. [:::interpro_reference:::]</accnos>
 <accnos>InterPro:IPR000887; KDPG and KHG aldolase. [:::interpro_reference:::]</accnos>
 <accnos>InterPro:IPR000483; Cysteine-rich flanking region, C-terminal. [:::interpro_refe
 <accnos>InterPro:IPR001611; Leucine-rich repeat. [:::interpro_reference:::]</accnos>
 <genomic_accnos>AFFYMETRIX:AFFY_MG_U74BV2:111295_at</genomic_accnos>
...
 <genomic_accnos>UNIGENE:Mm.85343</genomic_accnos>
 <comments>
 </comments>
 <references>
 </references>
 <members>
 </members>
 <groups>
<item type="Molecule" xlink:type="simple" xlink:href="molecule.xml#ID     (MO0000019394)" :
 </groups>
 <locations>
<go>GO: cellular_component: extracellular space; TAS; GO:0005615.</go>
</locations>
 <locations>
<go>GO: cellular_component: lipopolysaccharide receptor complex; ISS; GO:0046696.</go>
</locations>
```

Figura 2.4: Información de la molécula TLR4 mostrada como un fichero XML de TRANSPATH®.

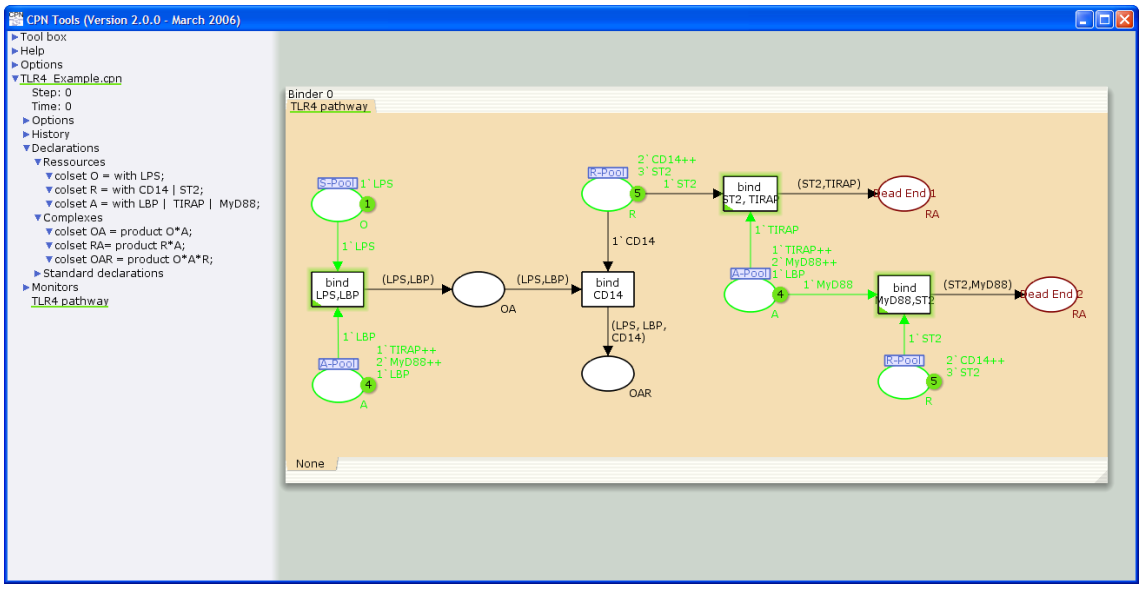


Figura 2.6: Reacciones del caso de estudirepresentadas en *CPN Tools*.

En [64] el método empleado para poder realizar la simulación a partir de los datos almacenados en TRANSPATH®es manual. Esto supone que el usuario/biólogo que va a realizar la simulación debe consultar de forma manual la base de datos obteniendo el listado de reacciones que intervienen en el *pathway* que desea estudiar. Con los datos obtenidos debe crear y editar de forma manual la correspondiente red de Petri coloreada en *CPN Tools* (figura 2.6), creando uno a uno cada uno de los lugares, transiciones, arcos, tokens, etc. En el trabajo al que se hace referencia (el estudio del pathway del TLR4 completo), esto supuso crear de forma manual —y únicamente para el ejemplo del *pathway* del TLR4— 75 lugares, 47 transiciones y casi un centenar de colores.

Capítulo 3

Ingeniería Dirigida por Modelos.

La evolución de la tecnología en el campo de la Ingeniería del Software ha permitido el desarrollo de sistemas cada vez más complejos. Esto en gran medida ha sido posible gracias a la introducción de técnicas que han posibilitado el incremento del nivel de abstracción en la descripción de problemas y sus soluciones. En la década de los 80 se dio un gran paso en este sentido mediante la aparición de las herramientas CASE (Computer Aided Software Engineering —Ingeniería de Software Asistida por Ordenador—), cuyo objetivo era dotar de métodos para el desarrollo de software creando herramientas que les dieran soporte. Estas herramientas permitían a los desarrolladores expresar sus diseños mediante representaciones gráficas, como diagramas de estructura o máquinas de estados, elevando el nivel de abstracción de la especificación de los sistemas software. No obstante esta tecnología no tuvo la aceptación que cabía esperar. El motivo hay que buscarlo en las limitaciones de los procesos de traducción que trasladaban las representaciones gráficas de los sistemas (mediante lenguajes gráficos de propósito general) a una plataforma o tecnología específica.

Los avances en el desarrollo de lenguajes de programación durante las pasadas dos décadas han conseguido elevar el nivel de abstracción en el desarrollo de software, aliviando los impedimentos de los primeros esfuerzos en la tecnología CASE. Los lenguajes basados en el paradigma de la orientación a objetos, como Java, C++ o C#, han dotado de una mayor expresividad en la codificación de sistemas, siendo su uso común en la mayor parte de los desarrollos, en detrimento de lenguajes estructurados clásicos, como Fortran o C. No obstante, la modificación y mantenimiento de los sistemas desarrollados se ha convertido en una tarea que implica un esfuerzo excesivo y tedioso.

La Ingeniería Dirigida por Modelos (Model Driven Engineering, MDE) tiene como objetivo organizar los niveles de abstracción y las metodologías de desarrollo, todo ello promoviendo el uso de modelos como artefactos principales a ser construidos y mantenidos. Un modelo está constituido por un conjunto de elementos que proporcionan una descripción sintética y abstracta de un sistema, concreto o hipotético. El término MDE fue propuesto por primera vez por Stuart Kent [37], en lo que se define como una marco general para

la especificación de los modelos y tareas de modelado necesarias para llevar a cabo un proyecto de desarrollo software desde principio a fin. Cualquier especificación puede ser expresada con modelos, y éstos pueden tener cualquier nivel de abstracción y expresar cualquier aspecto de un sistema. El proceso de desarrollo se convierte en un proceso de refinamiento y transformación entre modelos, de manera que el nivel de abstracción cada vez es menor, hasta que en un último paso se genera código para una plataforma específica. Un proceso MDE debe definir claramente la secuencia de modelos a desarrollar en cada nivel y describir cómo derivar a partir de un modelo un modelo de menor nivel de abstracción. El sistema a desarrollar es inicialmente descrito por un modelo que captura los requerimientos, independientemente de los detalles específicos de la plataforma o de cualquier decisión de implementación. Se trata de un modelo con el mayor nivel de abstracción posible, una descripción del problema a abordar.

La aplicación de las propuestas de MDE a las herramientas CASE solamente pasaba por un escollo: la ausencia de lenguajes de modelado y metodologías de desarrollo estándar que dieran soporte a los sistemas software en todo su ciclo de vida a través de estas herramientas. Además, la existencia de estándares permitiría una mayor interoperabilidad.

La Ingeniería Dirigida por Modelos es un campo en la Ingeniería del Software que, durante años, ha representado los artefactos software como modelos con el objetivo de incrementar la productividad, calidad, y reducir los gastos en el proceso de desarrollo de software. Recientemente, existe un interés creciente en este campo. Prueba de ello es la aproximación de Model Driven Architecture [47], apoyada por el OMG.

El Desarrollo Dirigido por Modelos ha evolucionado del campo de la Ingeniería Dirigida por Modelos. En él, no sólo las tareas de diseño y generación de código están involucradas, sino que también se incluyen las capacidades de trazabilidad, tareas de meta-modelado, intercambio y persistencia de modelos, etc. Para poder abordar estas tareas, las operaciones entre modelos, transformaciones, y consultas sobre ellos son problemas relevantes que deben ser resueltos. En el contexto de MDA se abordan desde el punto de vista de los estándares abiertos. En este caso, el estándar Meta Object Facility (MOF) [50], proporciona un mecanismo para definir metamodelos. El estándar Query/Views/Transformations (QVT) [49] indica cómo proporcionar soporte tanto para transformaciones como para consultas. A diferencia de otros lenguajes nuevos, QVT se apoya en el ya existente lenguaje *Object Constraint Language* (OCL) para realizar las consultas sobre los artefactos software.

Dentro de la ingeniería dirigida por modelos se ha propuesto una nueva disciplina denominada Gestión de Modelos. Ésta considera los modelos y las correspondencias entre ellos como entidades de primer orden, proporcionando un conjunto de operadores independientes de metamodelo y basados en teoría de conjuntos para tratar con ellos (Merge, Cross, Diff, ModelGen, etc.). Estos operadores proporcionan una solución reutilizable y componible para las tareas descritas anteriormente.

3.1. Estándares abiertos

Para dar respuesta a estos problemas en el contexto de MDE, el Object Management Group (OMG) [27] ha lanzado la iniciativa Model Driven Arquitecture (MDA) [47], como una aproximación a la especificación e interoperabilidad de sistemas basada en el uso de modelos formales. En MDA, los modelos independientes de la plataforma (platform-independent models, PIMs) son inicialmente expresados en un lenguaje de modelado independiente de la plataforma, como UML. El modelo independiente de la plataforma es traducido a un modelo específico para la plataforma considerada (platform-specific model, PSM), por ejemplo, la plataforma Java, usando reglas formales. Por último, y a partir del modelo específico para la plataforma, se genera el código del sistema en el lenguaje de programación objetivo (Java, C#,...). Además, se propone la automatización de las transformaciones entre modelos y de la generación de código, pudiendo centrar el proceso de desarrollo de software en las tareas de modelado.

MDA define un gran número de estándares de OMG, algunos de ellos explicados en mayor detalle en los siguientes apartados:

- *UML (Unified Modelling Language)*, que proporciona un vocabulario para describir gran cantidad de sistemas. UML se caracteriza por ser un vocabulario independiente de dominio, si bien tiene sus raíces en el modelado orientado a objetos.
- *CWM (Common Warehouse Metamodel)*, un vocabulario específico para el dominio de los sistemas relacionados con la minería o explotación de datos.
- *OCL (Object Constraint Language)*, un vocabulario que permite expresar consultas y restricciones sobre modelos. En una sección posterior nos centraremos en este lenguaje, en torno el cual gira el desarrollo de este proyecto.
- *QVT (Query/View/Transformation)*, un vocabulario que utiliza OCL para expresar transformaciones y relaciones de equivalencia sobre modelos.
- *XMI (XML Metadata Interchange)*, un vocabulario que permite el intercambio de modelos vía XML.
- *MOF (Meta Object Facility)*, es el metamodelo facilitado por MDA como vocabulario básico o metametamodelo.

3.1.1. MDA

Como hemos comentado, el grupo OMG ha propuesto un marco de trabajo en el ámbito de la ingeniería de modelos denominado MDA (Model Driven Architecture — Arquitectura dirigida por modelos). Éste pretende establecerse como un estándar «de facto» en este ámbito. MDA es un proceso de desarrollo de software. Por lo tanto el objetivo es producir sistemas informáticos ejecutables. La Arquitectura Dirigida por Modelos es un

marco de trabajo cuyo objetivo central es resolver el problema del cambio de tecnología e integración. La idea principal de MDA es usar modelos, de modo que las propiedades y características de los sistemas queden plasmadas de forma abstracta, y por tanto, los modelos no serían afectados por tales cambios.

Hay otros problemas que la MDA espera resolver [39] como: la baja productividad en los desarrollos de software y la interoperabilidad. El primero se solucionaría porque el análisis y desarrollo de los sistemas se hace a través de cambios a modelos de alto nivel, y a partir de dichos modelos, se procedería a la generación de código. La interoperabilidad se solucionaría porque la generación de código permitiría obtener código en diferentes tecnologías. Lo cual añade otra ventaja, la reutilización de los modelos.

MDA define dos clases de modelos: aquellos que son independientes de la plataforma (PIM o *Platform Independent Model*) y aquellos específicos de la plataforma (PSM o *Platform Specific Model*). Las definiciones de la guía de MDA [47] son las siguientes:

- *Plataforma.* Una plataforma es un conjunto de subsistemas y tecnologías que provee un conjunto coherente de funcionalidades a través de interfaces y unos patrones específicos de uso, los cuales pueden ser empleados por cualquier aplicación sin que ésta tenga conocimiento de los detalles de cómo la funcionalidad es implementada.
- *PIM.* Un modelo independiente de la plataforma es una vista de un sistema desde un punto de vista independiente de la tecnología.
- *PSM.* Un modelo específico de la plataforma es una vista de un sistema que depende de la tecnología donde se ejecutará el sistema.

El desarrollo de un sistema de acuerdo al marco de trabajo de MDA se inicia con la construcción de un PIM. Después el PIM es transformado en uno o más PSM. Por último, el código es generado a partir de los PSM. La operación fundamental en MDA es la transformación de PIM a PSM. La guía de MDA dice que los modelos PIM y PSM se expresan como modelos UML y que las transformaciones deben automatizarse al máximo posible.

En [43] se clasifican las transformaciones en dos tipos: vertical y horizontal. Es horizontal si el modelo origen y el modelo destino pertenecen al mismo nivel de abstracción, en MDA es una transformación de PIM a PIM o de PSM a PSM. Una transformación es vertical si el modelo origen y el modelo destino están situados en distintos niveles de abstracción. En MDA es una transformación de PIM a PSM o de PSM a código. Desde el punto de vista de MDE también interesan las transformaciones de PIM a PIM y de PSM a PSM (de acuerdo a la clasificación de modelos de MDA). Otras clasificaciones de modelos posibles, proporcionan otro conjunto de transformaciones de modelos de interés. En MDA la transformación esencial es de PIM a PSM, de abstracto a menos abstracto. En MDE esa es sólo una de las posibles transformaciones de modelos.

Las versiones iniciales de MDA sirvieron como base de MDE, que generalizó MDA, y que define las transformaciones en el contexto del metamodelado. En MDE la forma más

aceptada de definir los modelos es a través del metamodelado, y las transformaciones a través de lenguajes de transformación de modelos. En cambio, en la propuesta original de MDA el metamodelado no es una condición necesaria. Las versiones posteriores de MDA, incorporaron después las ideas de MDE, que dieron lugar a MOF y QVT.

3.1.2. MOF

MOF (Meta Object Facility) es el metamodelo facilitado por MDA como vocabulario básico o metametamodelo. Mediante MOF pueden definir nuevos metamodelos, y por lo tanto nuevos vocabularios (de hecho se podría decir lenguajes, pero es conveniente no utilizar el término para evitar confusiones) con las mismas herramientas con que se definen modelos. Por otra parte, cabe preguntarse si existe un vocabulario de modelos superior que se utiliza para definir metamodelos.

La respuesta es que sí, a este metamodelo de metamodelos se le denomina metametamodelo. Pero como también es un modelo, ¿se podría seguir extendiendo esta pirámide de forma infinita?

En la práctica esto no tiene sentido, y los metamodelos y modelos se suelen organizar en una estructura de cuatro capas M3-M0 con la siguiente distribución:

- En el nivel M1 se sitúan los modelos, tal y como los hemos introducido aquí, descripciones abstractas de un sistema
- En la capa inmediatamente superior, denominada M2, se sitúan los metamodelos, «vocabularios para definir modelos».
- El nivel M3, que cierra la estructura por arriba, contiene el vocabulario base que permite definir metamodelos. Cabe resaltar que este nivel suele contener un único vocabulario, que caracteriza la aproximación de modelos escogida.

  Es imperativo que este vocabulario o metametamodelo esté definido utilizando como vocabulario a sí mismo, de ahí que se cierre la estructura.
- El nivel inferior, denominado M0, es en el que se sitúan los datos, es decir las instancias del sistema bajo estudio.

Esta estructura de cuatro capas (representada en la figura 3.1) permite conseguir una gran riqueza de vocabularios para describir distintos tipos de sistemas, o bien para proporcionar diversos puntos de vista de un mismo sistema.

Resulta interesante destacar que esta asignación fija de niveles puede resultar confusa en ocasiones. Quizá es más interesante fijar como idea fundamental la relación entre un modelo y su vocabulario, y darse cuenta de que esta relación ocurre en todos los niveles descritos. Esta relación se denomina informalmente «relación instancia-de». Decimos que

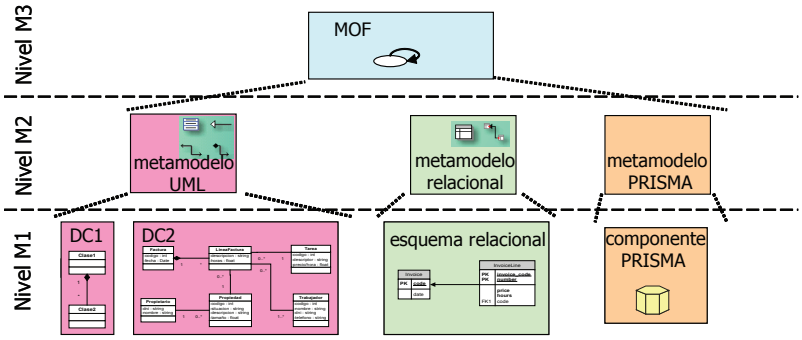


Figura 3.1: Arquitectura de niveles de MOF.

un modelo «x» es una instancia de un vocabulario «x+1», al que denominamos metamodelo. El modelo está en el nivel inferior, nivel de instancia; y el metamodelo en el superior, nivel meta. Podemos aplicar esta dualidad al metamodelo «x+1» ya que si ahora lo situamos en el nivel instancia, vemos que también «x+1», necesariamente, está definido por un vocabulario «x+2». Por lo tanto podemos situar un modelo tanto en el nivel meta y decir que tiene instancias, como en el nivel instancia, y decir que proviene de un metamodelo. Cabe resaltar el caso especial del metametamodelo (nivel M3) que se define a sí mismo, por lo tanto se podría decir que es una instancia de sí mismo.

MDA sitúa en la capa M2 diversos metamodelos bien conocidos que están definidos mediante MOF, como por ejemplo UML[48], CWM o QVT[53].

3.1.3. OCL

Object Constraint Language (OCL) es un lenguaje notacional para el análisis y diseño de sistemas software. Está definido como lenguaje estándar para dar soporte al Unified Modeling Language (UML), el cual es un estándar de OMG para el análisis y diseño orientado a objetos. OCL da soporte a UML para la especificación de restricciones y consultas sobre modelos, permitiendo definir y documentar los modelos UML de forma más precisa.

Un modelo UML, como por ejemplo un diagrama de clases o un diagrama de estados, no constituye una especificación lo suficientemente precisa y libre de ambigüedades de un sistema. Las expresiones OCL completan esta especificación, convirtiéndose en información vital para los modelos orientados a objetos y otros objetos para el modelado. Esta información a menudo no puede ser expresada mediante un diagrama.

Cada expresión OCL se refiere a tipos (por ejemplo, clases, interfaces,. . .) definidos en diagramas UML. De esta manera, una expresión OCL estará siempre ligada a un diagrama UML (no tendría sentido de manera aislada).

3.1.3.1. Características de OCL

En UML 1.1, OCL era un lenguaje para expresar obligaciones (constraints) sobre elementos de un modelo, definidas como restricciones sobre uno o mas valores de un modelo orientado a objetos o sistema.

En UML 2 se incluye información adicional, como consultas, valores de referencia, condiciones de estado, reglas de negocio, etc. En definitiva, se puede utilizar OCL para expresar cualquier expresión sobre elementos de un diagrama.

Las expresiones OCL pueden ser usadas en cualquier punto de un modelo para indicar un valor. Un valor puede ser un valor simple, como un entero, pero puede ser también la referencia a un objeto, una colección de valores, o una colección de referencias a objetos. Una expresión OCL puede representar, por ejemplo, un valor booleano usado en la condición de un diagrama de estados, o un mensaje en un diagrama de interacción. Una expresión OCL puede referirse a un objeto específico en un diagrama de interacción o de objetos. Por ejemplo, la siguiente expresión define el cuerpo de la operación asientosDisponibles() de la clase Vuelo:

```
1       context Vuelo::asientosDisponibles() : Integer
2       body: self.avion.tipo.cant_de_asientos - self.pasajeros->size()
```

Listado 3.1: Ejemplo de expresión OCL: cuerpo de la operación asientosDisponibles().

OCL está basado en la teoría de conjuntos y lógica de predicados, y tiene una semántica matemática formal [60]. Su notación, sin embargo, no utiliza símbolos matemáticos. De esta manera, OCL proporciona el rigor y precisión de un lenguaje formal y la facilidad de uso de un lenguaje natural.

Una característica esencial de OCL es que es un lenguaje tipado. Las expresiones OCL son usadas para modelar y especificar. Debido a que muchos modelos no son ejecutables directamente, muchas expresiones OCL estarán reflejadas aún cuando no existan versiones ejecutables del sistema. Sin embargo, debe ser la posible la comprobación de la corrección de estas expresiones, sin tener que producir una versión ejecutable del modelo. Como lenguaje tipado, las expresiones OCL pueden ser comprobadas durante el modelado, antes de la ejecución. De esta manera, los errores del modelo pueden ser eliminados en etapas tempranas.

Otro aspecto esencial de OCL es que se trata de un lenguaje declarativo. En los lenguajes procedurales, como son la mayoría de los lenguajes de programación, las expresiones son descripciones de las acciones que se quieren llevar a cabo. En un lenguaje declarativo, una expresión establece lo que debería hacerse, pero no cómo. Para asegurar esto, las expresiones OCL no tienen efectos laterales. Esto quiere decir que una expresión OCL no puede cambiar el estado del sistema.

El modelador puede tomar decisiones a un alto nivel de abstracción. Por ejemplo, en la definición del cuerpo de la expresión de la operación asientosDisponibles() del ejemplo

anterior, se especifica aquello que la operación debe calcular, pero no se establece cómo debe hacerse. El cómo dependerá de la estrategia que se siga en la implementación del sistema.

3.1.4. QVT

Para soportar transformaciones se ha seguido el estándar Query View Transformation (QVT) [53] propuesto por el OMG. La especificación de QVT depende de otros dos estándares de la OMG como son MOF 2.0 y OCL 2.0. De esta manera, la utilización de la especificación de QVT para especificar transformaciones, aporta reutilización de tecnología que sigue estándares y reducción de la curva de aprendizaje de la herramienta.

La especificación QVT se define a través de dos dimensiones ortogonales: la dimensión del lenguaje y la dimensión de interoperabilidad, cada una de las cuales tiene una serie de niveles. La intersección de niveles de las dos dimensiones define un punto de compatibilidad QVT (QVT - compliance).

La dimensión del lenguaje define los diferentes lenguajes de transformación presentes en la especificación QVT. Concretamente son tres: Relations, Core y Operational, y la principal diferencia entre ellos es su naturaleza declarativa o imperativa.

En la dimensión de la interoperabilidad se encuentran aquellas características que permiten a una herramienta que cumple el estándar QVT interoperar con otras herramientas. Concretamente, son cuatro: sintaxis ejecutable, XMI ejecutable, sintaxis exportable y XMI exportable.

- La sintaxis ejecutable se traduce en una implementación que facilite la importación o lectura, y posterior ejecución de una sintaxis concreta que describa una transformación definida en el lenguaje dado por la dimensión del lenguaje.
- XMI ejecutable dictamina que una implementación debe facilitar la importación o lectura, y posterior ejecución de una serialización XMI de una transformación que conforma con el metamodelo de MOF del lenguaje dado por la dimensión del lenguaje.
- La sintaxis exportable establece que una implementación debe proporcionar facilidad para exporta una transformación entre modelos en la sintaxis concreta del lenguaje dado por la dimensión del lenguaje.
- XMI exportable es el nivel de interoperabilidad que debe facilitar la exportación de transformaciones entre modelos como serializaciones XMI que conformen con el metamodelo MOF del lenguaje dado por la dimensión del lenguaje.

3.1.4.1. Lenguajes del estándar QVT

La especificación QVT tiene una naturaleza híbrida: declarativa e imperativa, con la parte declarativa dividida en una arquitectura de dos niveles que formará el framework

para la ejecución semántica de la parte imperativa.

La parte declarativa de esta especificación está estructurada en una arquitectura de dos capas:

- Un metamodelo Relations conocido, y un lenguaje que soporta pattern matching de objetos complejos y creación de plantillas de objetos. La trazabilidad entre los elementos del modelo involucrados en la transformación se crea implícitamente.
- Un metamodelo Core, y un lenguaje definido usando extensiones mínimas de EMOF y OCL. Todas las clases de trazabilidad están definidas explícitamente como modelos de MOF, y la creación y eliminación de una instancia de clase de trazabilidad se define de la misma manera que la creación y eliminación de cualquier otro objeto.

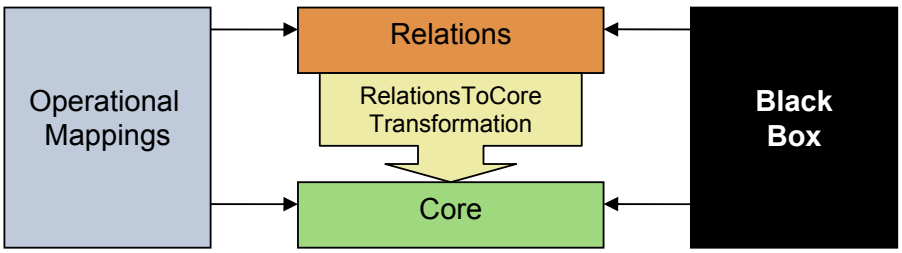


Figura 3.2: Relaciones entre los metamodelos QVT.

El lenguaje *Relations*

El lenguaje Relations proporciona una especificación declarativa de relaciones entre modelos MOF. Soporta pattern matching de objetos complejos y de manera implícita crea clases de trazabilidad y sus instancias correspondientes, de forma que permite guardar todo lo que ocurre durante la ejecución de una transformación.

El lenguaje *Core*

Este es un lenguaje/modelo que solo soporta *pattern-matching* sobre un conjunto de variables, evaluando condiciones sobre variables de un conjunto de modelos. Core trata a todos los elementos del modelo origen, destino y de trazabilidad de manera simétrica y su potencia es equiparable a la del lenguaje Relations. Esto se debe, a que por su relativa simplicidad su semántica puede ser definida de manera más sencilla, aunque la definición de transformaciones sea más verbosa. Además, los modelos de trazabilidad han de ser definidos de manera explícita y no pueden deducirse de la definición de la transformación como ocurría en el lenguaje Relations.

El modelo Core debe ser implementado directamente, o simplemente usado como referencia para la semántica del Relations, la cual está mapeada a Core a través de una transformación.

Analogía de máquina virtual

Es posible trazar una analogía con la arquitectura Java™, donde el lenguaje *Core* es como *Java Byte Code* y la semántica de *Core* se comporta como la máquina virtual Java. El lenguaje Relations desempeña el rol del lenguaje Java, y la transformación estándar de Relations a Core es la especificación de un compilador Java que genera Byte Code.

El lenguaje *Operational Mappings*

Este lenguaje está especificado como estándar para proporcionar implementaciones imperativas. Proporciona extensiones OCL con efectos laterales que da un mayor estilo procedural y una sintaxis más cercana a la programación imperativa.

Este lenguaje define operaciones de mappings que pueden ser usadas para implementar una o más Relations a partir de una especificación Relations, cuando sea difícil proporcionar una especificación declarativa más pura. Estas operaciones pueden invocar otras operaciones de mappings con el objetivo de crear modelos de trazabilidad entre elementos de modelos.

Es posible escribir transformaciones completas utilizando operaciones de mapping, denominándose este tipo de transformaciones: transformaciones operacionales.

Implementación Black-box

Las operaciones MOF deben poder derivarse de Relations haciendo posible integrarlas con cualquier implementación de una operación MOF con la misma signatura. Esto resulta beneficioso por las siguientes razones:

- Permite codificar algoritmos complejos en cualquier lenguaje de programación con correspondencia a MOF.
- Permite el uso de librerías específicas de dominio para calcular ciertas propiedades del modelo. Por ejemplo, en matemáticas, en ingeniería y en otros muchos campos existen grandes librerías que almacenan algoritmos específicos de dominio, que serían muy difíciles de expresar usando OCL.
- Permite ocultar la implementación de algunas partes de una transformación.

Sin embargo, esta integración puede resultar peligrosa puesto que la implementación asociada a esta integración tiene libre acceso a las referencias de objetos en los modelos. Las implementaciones Black-box no tienen una relación implícita hacia Relations, y cada

Black-box debe implementar explícitamente una relación que sea responsable de conservar la trazabilidad entre los elementos del modelo relacionado con la implementación de la operación MOF.

En aras de extender la analogía con la arquitectura Java, la habilidad para invocar implementaciones «Black-Box» y «Operational Mappings», puede considerarse equivalente con la llamada a la Java Native Interface (JNI).

3.1.4.2. El lenguaje *Relations*

Transformaciones y tipos de modelos

En el lenguaje Relations, una transformación entre modelos candidatos se especifica como un conjunto de relaciones que han de tener lugar para llevar a cabo la transformación. Un modelo candidato es cualquier modelo que conforme a un tipo de modelo o metamodelo, que es una especificación de los diferentes tipos que pueden tener los elementos del modelo que lo conforma. Los modelos candidatos tienen un nombre y los tipos de los elementos que pueden contener, los cuales quedan restringidos al paquete al que se hace referencia en la declaración del modelo candidato. Por ejemplo:

```
1       transformation uml2Rdbms (uml : SimpleUML, rdbms : SimpleRDBMS)
2       }
```

Listado 3.2: Declaración de una tranformación en QVT-Relations.

En esta declaración de transformación llamada «uml2Rdbms», hay dos modelos candidatos: «uml» y «rdbms». El modelo «uml» se declara referenciando el paquete «SimpleUML» como su metamodelo, y el modelo «rdbms» hace lo propio con el paquete «SimpleRDBMS». Una «transformation» puede invocarse tanto para comprobar la consistencia de dos modelos como para transformar un modelo en otro.

Dirección de ejecución de una transformación

Una «transformation» invocada para realizar una transformación se ejecuta en una dirección determinada seleccionando uno de los modelos candidatos como modelo destino u objetivo. El modelo destino debe estar vacío o contener elementos para ser relacionados por la transformación. El procedimiento de transformación se basa en comprobar que todas las relaciones definidas en la transformación se cumplen y para aquellas relaciones que no se cumplen, se intenta cumplirlas creando, eliminando o modificando el modelo destino.

Relaciones y dominios

Las relaciones en una transformación definen restricciones que deben satisfacer los elementos de los modelos candidatos. Una relación o relation definida para dos o más dominios, y un par de predicados when y where, especifica una relación que se debe satisfacer

entre los elementos de los modelos candidatos.

Un dominio o domain es una variable tipada que puede hacer matching en un modelo de un metamodelo dado. Un dominio tiene un patrón que puede ser visto como un grafo de nodos de objetos, cuyas propiedades y links de asociación se originan de una instancia del tipo del dominio. Alternativamente, un patrón puede ser considerado como un conjunto de variables y un conjunto de restricciones, que los elementos del modelo vinculados a esas variables deben satisfacer para logar una correspondencia válida de ese patrón. Un patrón de dominio o domain pattern puede considerarse como una plantilla para objetos, y sus propiedades han de ser localizadas, modificadas o creadas en un modelo candidato para satisfacer la relación.

En el siguiente ejemplo, se declaran dos dominios que corresponderán a elementos de los modelos «uml» y «rdbms». Cada dominio especifica un patrón simple, un paquete con un nombre y un esquema con un nombre. Ambas propiedades de «nombre» se corresponden con la misma variable «pn», lo que implica que deberían tener el mismo valor:

```
1       relation PackageToSchema
2       {
3          domain uml p: Package {name=pn}
4          domain rdbms s: Schema {name=pn}
5       }
```

Listado 3.3: Declaración de una relation en QVT-Relations.

Cláusulas when y where

Como se muestra en el siguiente ejemplo de la relación «ClassToTable», una relación puede estar limitada por dos conjuntos de predicados, una cláusula when y una cláusula where. La cláusula when define las condiciones bajo las cuales se necesita satisfacer la relación, por lo que la relación «ClassToTable» se cumplirá solo cuando se cumpla la relación «PackageToSchema», es decir, cuando el paquete contiene la clase y el esquema contiene la tabla. La cláusula where define la condición que deben satisfacer todos los elementos de todos los modelos que participan en la relación, y puede restringir cualquiera de las variables en la relación y sus dominios. Por tanto, siempre que se cumpla la relación «ClassToTable», la relación «AttributeToColumn» también se cumplirá.

```
1       relation ClassToTable
2       {
3          domain uml c:Class {
4                   namespace = p:Package { },
5                   kind = 'Persistent',
6                 name = cn
7       }
8
9          domain rdbms t:Table {
10                  schema = s:Schema { },
```

```
                name = cn,
                column = cl:Column {
                     name = cn+'_tid',
                     type='NUMBER' },
                 primaryKey = k:PrimaryKey {
                     name = cn+'_pk',
                     column = cl }
        }
        when {
           PackageToSchema(p,s);
        }
        where {
           AttributeToColumn(c,t);
        }
    }
```

Listado 3.4: Declaración de cláusulas *when* y *where* en QVT-Relations.

Las cláusulas when y where pueden contener cualquier expresión OCL además de las expresiones de invocación de la relación.

La invocación de relaciones permite componer relaciones complejas a partir de relaciones más simples.

Relaciones Top-level

Una transformación contiene dos tipos de relaciones: las top-level y las no-top-level. La ejecución de una transformación requiere que todas sus relaciones top-level se cumplan, mientras que las relaciones no-top-level solo han de ser satisfechas cuando son invocadas directa o transitivamente desde la cláusula where de otra relación.

```
    Transformation uml2Rdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {
          top relation PackageToSchema { . . . }
          top relation ClassToTable { . . . }
          relation AttributeToColumn { . . . }
    }
```

Listado 3.5: Declaración de relaciones *top-level* en QVT-Relations.

Una relación top-level tiene la palabra clave «top» que la distingue sintácticamente. En el ejemplo anterior, tanto «PackageToSchema» como «ClassToTable» son relaciones top-level mientras que «AttributeToColumn» es una relación no-top-level.

Check y Enforce

El que una relación sea de transformación o no viene determinado por el dominio destino, que estará marcado como checkonly o enforce. Cuando una transformación se satisface en la dirección de un dominio checkonly, simplemente se comprueba si existe una correspondencia válida en el modelo relevante que satisfaga la relación. Cuando una

transformación se ejecuta en la dirección de un dominio enforce, si la comprobación falla, se modifica el modelo destino tanto como sea necesario para que satisfaga la relación.

En el ejemplo siguiente, el dominio para el modelo «uml» está marcado como checkonly y el dominio para el modelo «rdbms» está marcado como enforce.

```
1       relation PackageToSchema {
2          checkonly domain uml p: Package { name=pn }
3          enforce domain rdbms s: Schema  { name=pn }
4       }
```

Listado 3.6: Declaración de dominion *checkonly* y *enforce* en QVT-Relations.

Si estuviéramos ejecutando en la dirección de «uml» y existiera un esquema en «rdbms» para el cual no tenemos un paquete correspondiente con el mismo nombre en «uml», se detectaría una inconsistencia pero no se crearía el paquete puesto que el modelo «uml» es checkonly y no enforce.

Sin embargo, si estuviéramos ejecutando la transformación «uml2Rdbms» en la dirección de «rdbms», entonces para cada paquete en el modelo de «uml», la relación primero comprobaría si existe un esquema con el mismo nombre en el modelo «rdbms», y si no es así, se crearía un nuevo esquema en ese modelo con el nombre dado. Para considerar una variación del caso propuesto, si ejecutamos la transformación en la dirección de «rdbms» y no hay un paquete con el mismo nombre que en «uml», entonces el esquema se eliminará del modelo «rdbms» para forzar la consistencia en el dominio enforce.

La aplicación de las reglas depende únicamente del dominio destino.

Pattern-Matching

Los patrones o *patterns* asociados a los dominios reciben el nombre de «object template expressions». Para ilustrar cómo tiene lugar el matching de estos patrones utilizaremos el ejemplo del «ClassToTable» utilizado con anterioridad.

«ClassToTable» define varias expresiones de tipo «object template expression» que se utilizarán para hacer pattern matching en los modelos candidatos. Por ejemplo, el siguiente «object template expression» está asociado al dominio de «uml»:

```
1       c: Class {
2                   namespace = p: Package { },
3                   kind = 'Persistent',
4                   name = cn
5                 }
```

Listado 3.7: *Pattern-matching* en QVT-Relations.

Las combinaciones resultantes de un «object template expression» unen elementos de un modelo candidato con variables declaradas en el dominio. Un «object template expression» debe efectuarse en un contexto donde algunas de las variable del domino estén

enlazadas con elementos de un modelo. En este caso el «object template expression» solo encuentra enlaces para tres variables del dominio.

El pattern matching ligará todas las variables de la expresión («c», «p», y «cn»), empezando desde la variable raíz del domino «c» de tipo «Class». En este ejemplo, la variable «p» debería también estar ligada como resultado de la evaluación de la expresión «PackageToSchema(p,s)» en la cláusula when. El proceso de matching pasa por filtrar todos los objetos de tipo «Class» en el modelo «uml», eliminando cualquier objeto que no tenga los mismos valores literales que el «object template expression». En el ejemplo, cualquier «Class» con su propiedad «kind» distinta de «Persistent» es eliminada.

Para propiedades que se comparan con variables, como «name=cn», aparecen dos casos. Si la variable «cn» tiene un valor asignado, entonces cualquier «Class» que no tenga el mismo valor para su propiedad «name» será eliminada. Si la variable «cn» es libre, como sucede en el ejemplo, se le asignará el valor de la propiedad del nombre de todas las clases que han sido eliminadas filtrando, debido a una incompatibilidad con otras comparaciones de la propiedad.

Entonces el matching avanza a propiedades cuyos valores son comparados con «object template expressions» anidadas. Por ejemplo, la propiedad «namespace = p: Package » hará matching solo con aquellas clases cuya propiedad «namespace» tenga una referencia no nula a un «Package». Al mismo tiempo, la variable «p» apuntará al «Package». Sin embargo, puesto que en nuestro ejemplo «p» toma valor en la cláusula when, solo se hará pattern matching con aquellas clases cuyas propiedades «namespace» tengan una referencia al mismo paquete al que se refiere «p».

Se permite el anidamiento profundo de «object template expressions», y el matching y la toma de valores de las variables se realiza recursivamente hasta que haya una serie de tuplas de valores que correspondan a variables de un dominio y su «object template expression». Por ejemplo las tres variables: «c», «p» y «cn», crean una 3-tupla, y cada match válido dará como resultado una única tupla que represente la unión de un valor con una variable.

En una invocación a una relación dada, puede haber varios matchings válidos para una «template expression», la manera de procesar esta multiplicidad depende de la dirección en la que ejecutemos la relación.

Por ejemplo, si se ejecuta la relación «ClassToTable» con «rdbms» como modelo destino, entonces por cada matching válido del dominio «uml», tendría que existir al menos un matching válido del dominio «rdbms» que satisfaga la cláusula where. Si para un matching válido del dominio «uml», no existe un matching válido del dominio «rdbms», entonces puesto que el dominio «rdbms» es enforce, se crean objetos y las propiedades se ponen en la «template expression» asociada con el dominio «rdbms». También, para cada matching válido del dominio «rdbms», tendría que existir al menos un matching válido del dominio «uml» que satisfaga la cláusula where (esto exige que el dominio «uml» esté marcado como checkonly); en otro caso los objetos se eliminarán del modelo «rdbms» por lo que no seguirá siendo un matching válido.

Keys y creación de objetos usando patrones

Como se mencionó anteriormente, un «object template expression» también sirve como plantilla para crear un objeto en un modelo destino. Cuando para un matching válido dado de un patrón de dominio origen, no existe un matching en el patrón del dominio destino, entonces se utiliza el «object template expression» del dominio destino como plantilla para crear objetos en el modelo destino. Por ejemplo, cuando se ejecuta «ClassToTable» con «rdbms» como modelo destino, el siguiente «object template expression» sirve como plantilla para crear objetos en el modelo «rdbms»:

```
1       t: Table {
2          schema = s: Schema { },
3          name = cn,
4          column = cl: Column { name=cn+'_tid', type='NUMBER' },
5          primaryKey = k: PrimaryKey { name=cn+'_pk', column=cl }
6       }
```

Listado 3.8: Creación de objetos usando patrones en QVT-Relations.

La plantilla asociada con «Table» especifica que un objeto de este tipo debe crearse con las propiedades «schema», «name», «column» y «primaryKey» inicializadas a los valores de la «object template expression». Análogamente, las plantillas asociadas con «Column», «PrimaryKey», etc, especifican como deben crearse sus respectivos objetos.

Cuando se creen objetos, se debe asegurar que no se creen objetos duplicados cuando los objetos requeridos ya existen. Sin embargo, en ocasiones, simplemente se querrá actualizar objetos existentes, lo que deriva en un problema sobre cuándo actualizar objetos existentes o no permitir la creación de un objeto cuando éste ya existe con el fin de evitar objetos duplicados. MOF permite etiquetar como identificador una sola propiedad de una clase, sin embargo, para muchos metamodelos esto resulta insuficiente. El metamodelo Relations introduce el concepto de Key, que define un conjunto de propiedades de clase que identificarán de manera única el objeto instancia de una clase del modelo. Una clase puede tener múltiples keys al igual que ocurre en las bases de datos relacionales.

Por ejemplo, continuando con el ejemplo de la relación «ClassToTable», queremos especificar que en los modelos de tipo «simpleRDBMS», una tabla está identificada únicamente por dos propiedades: su nombre y el esquema al que pertenece.

```
1       key Table { schema, name };
```

Listado 3.9: Declaración de *Keys* en QVT-Realations.

Las keys se utilizan en el tiempo de la creación de objetos si un «object template expression» tiene las propiedades que corresponden a una key de la clase asociada, entonces las keys se utilizan para localizar un matching de objeto en el modelo; se crea un objeto solo cuando no existe un matching de objeto.

En el ejemplo, tomando el caso en el que se tiene una clase persistida llamada «foo» en el modelo «uml», y existe una tabla con un matching de nombre «foo» en un matching de esquema en el modelo «rdbms» pero la tabla no tiene matching de valores con las propiedades «column» y «primaryKey». En este caso, de acuerdo con el matching semántico de patrones, el modelo «rdbms» no tiene un matching válido para el patrón asociado a la «Table», por lo que es necesaria la creación de objetos que cumplan esa relación. Sin embargo, desde que la tabla existente hace matching con las propiedades especificadas por la key, no es necesario crear una nueva tabla, tan solo hay que actualizar las propiedades «column» y «primaryKey» de la tabla.

Ejecución de una transformación en modo *checkonly*

Una transformación puede ser ejecutada en modo checkonly. En este modo, la transformación simplemente comprueba si las relaciones se cumplen en todas las direcciones. No se realiza ningún forzado en ninguna dirección, independientemente de cómo estén marcados los dominios, checkonly o enforce.

Sintaxis y semántica abstracta

Paquete *QVTBase*

Este paquete contiene un conjunto de conceptos básicos, muchos reutilizados de las especificaciones de EMOF y OCL, que estructuran transformaciones, sus reglas y sus modelos de entrada y salida. También introduce la noción de patrón, o en inglés pattern, como un conjunto de predicados sobre variables en expresiones OCL. Estas clases se extienden en paquetes de lenguajes específicos para proporcionar semánticas de lenguajes específicos.

Transformation Una transformación o en inglés transformation, define como un conjunto de modelos pueden ser transformados en otro. Ésta contiene un conjunto de reglas, o en inglés rules, que especifican el comportamiento de su ejecución. Se ejecuta sobre un conjunto de modelos cuyos tipos están especificados por un conjunto tipos de modelos o metamodelos asociados con la transformación.

Sintácticamente, una transformación es una subclase de «Package» y de «Class». Como «Package» proporciona un espacio de nombres para las reglas que contiene; como «Class» puede definir propiedades y operaciones-propiedades para especificar valores de configuración.

TypedModel Un tipo de modelo o en inglés TypedModel, especifica un parámetro nominado y tipado de una transformación. En ejecución, un modelo que se pasa como parámetro a una transformación, solo puede contener aquellos elementos cuyos tipos están especificados en el conjunto de paquetes de modelos asociados con el tipo de modelo. Una transformación, siempre es ejecutada en una determinada dirección, seleccionando uno de los tipos de modelo como objetivo o destino de la transformación.

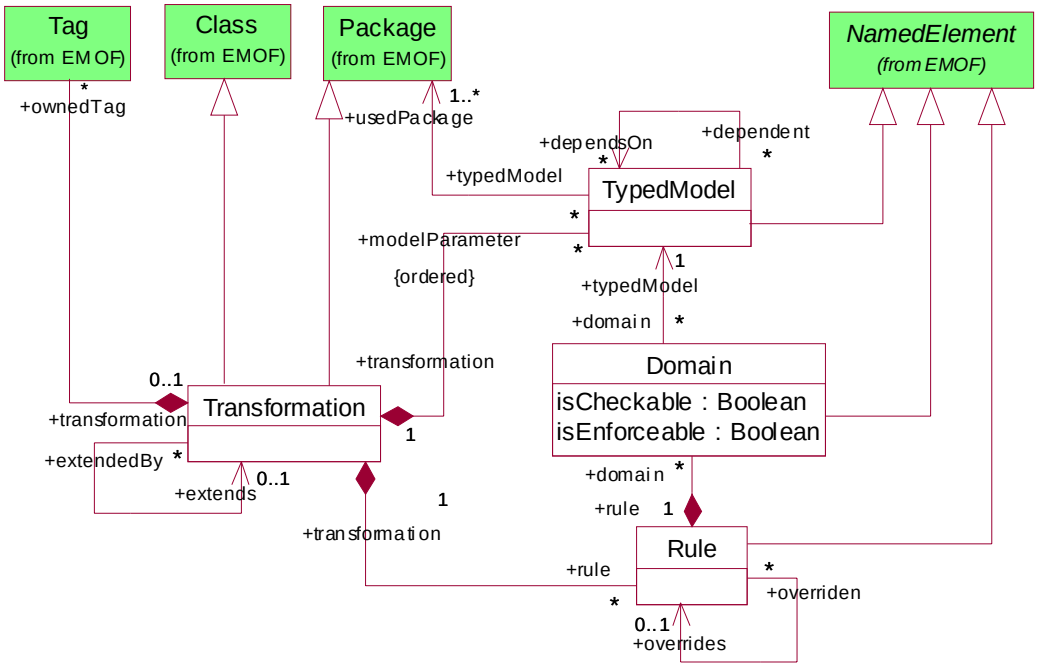


Figura 3.3: Paquete QVTBase - Transformaciones y reglas.

Domain Un domain o dominio especifica un conjunto de elementos de modelo de un tipo de modelo que son de interés para una regla. Domain es una clase abstracta, cuyas subclases concretas son responsables de definir el mecanismo exacto de especificación, por el cual han de especificarse el conjunto de elementos de modelo de un dominio.

Un domino debe estar marcado como checkonly o enforce. Un dominio de tipo checkonly declara que su regla solo es requerida para comprobar, que los elementos de modelos especificados por el dominio existen en el modelo destino, y devolver errores cuando no lo son. Un dominio de tipo enforce declara que su regla debe asegurarse, y forzar cuando sea necesario, que los elementos de modelo especificados por el dominio existan en el modelo destino, cuando la transformación se ejecute en la dirección del tipo de modelo asociado con el dominio.

Rule Una regla o en inglés rule, especifica como los elementos de modelos especificados por sus dominios son relacionados con otros, y como han de computarse por elementos de modelos de otros dominios. Rule es una clase abstracta cuyas subclases concretas son responsables de especificar la semántica de cómo se relacionan los dominios y cómo son computados por otros.

Una regla puede sobrescribir otra regla de manera condicional. La regla que sobrescribe se ejecuta en lugar de la sobrescrita cuando se satisfacen las condiciones de

sobrescritura.

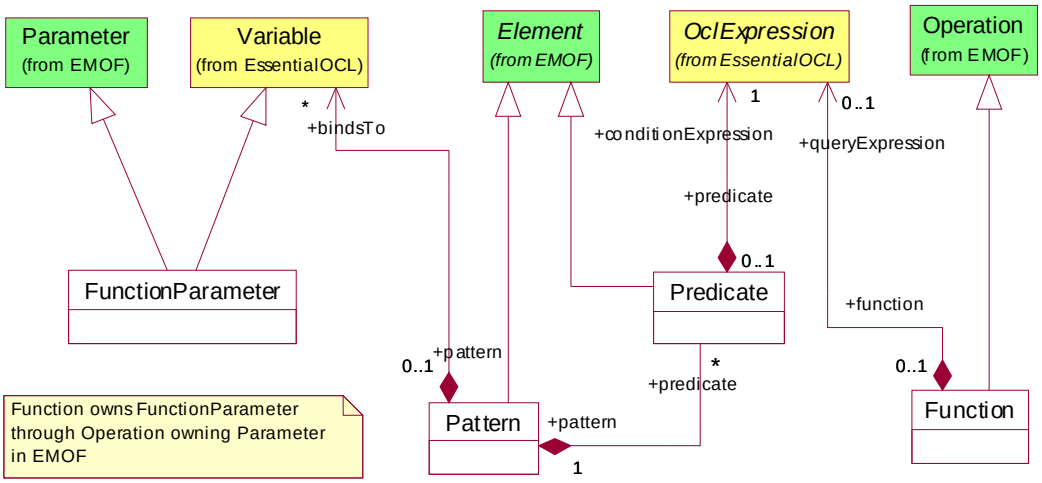


Figura 3.4: Paquete QVTBase - Patrones y funciones.

Function Una function o función, es una operación libre de efectos laterales contenida en una transformación. Una función debe producir el mismo resultado siempre que sea invocada con los mismos argumentos. Una función puede especificarse mediante una expresión OCL, o mediante una implementación black-box.

FunctionParameter Un function parameter o parámetro de función, especifica el parámetro de una función.

Sintácticamente, es una subclase de las clases «Parameter» y «Variable». Por ser subclase de «Variable», permite a las expresiones OCL que especifican una función acceder a los parámetros como variables nominadas.

Predicate Un predicado o en inglés predicate, es una expresión «booleana» contenida en un patrón. Está especificado por una expresión OCL que contiene referencias a las variables del patrón que tiene el predicado.

Pattern Un patrón o en inglés pattern, es un conjunto de variables y predicados, que cuando son evaluados en el contexto de un modelo, dan como resultado un conjunto de valores para las variables.

Paquete QVTTemplate

TemplateExp Una template expression o plantilla de expresión, especifica un patrón que puede combinarse con elementos de modelo en un modelo candidato de una transformación. El elemento de modelo combinado debe estar ligado a una variable

y a su vez, esta variable debe ser usada en otras partes de la expresión. Solo se produce el matching con elementos de modelo cuando la expresión where asociada con la template expression se evalúa a true. Una template expression debe poder hacer matching tanto con un solo elemento de modelo, como con una colección de elementos de modelo.

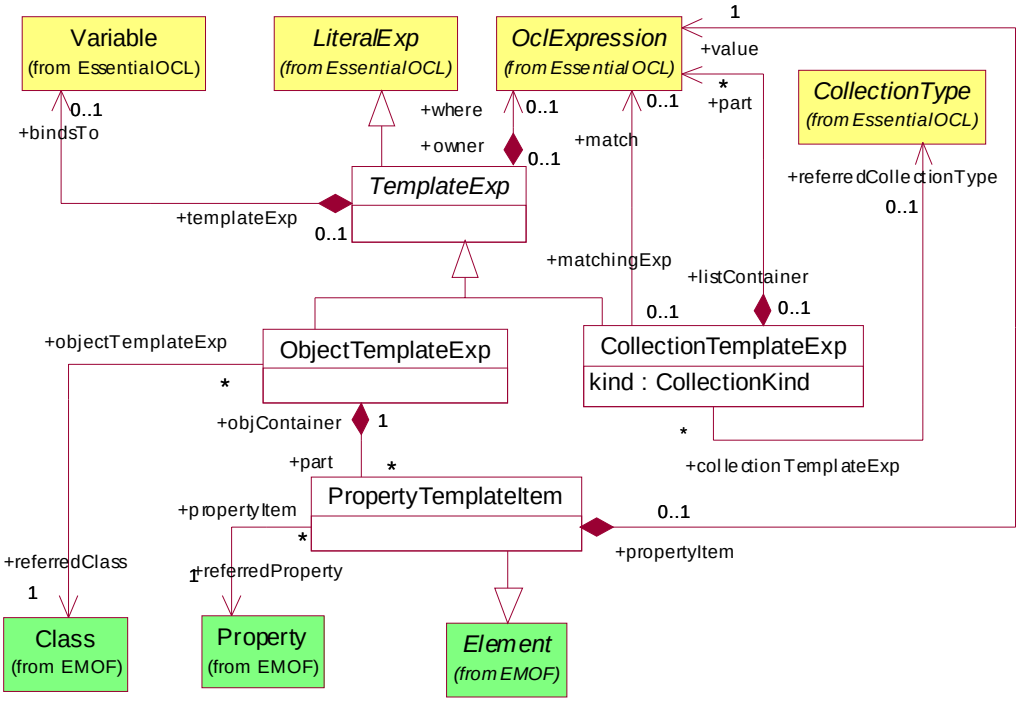


Figura 3.5: Paquete QVTTemplate

ObjectTemplateExp Una object template expression, especifica un patrón que debe poder hacer matching solo con un elemento de modelo, y tiene un tipo especificado por la clase referida. Se especifican mediante una colección de «property template ítems», cada uno de los cuales se corresponde con atributos diferentes de la clase referida.

CollectionTemplateExp Una collection template expression especifica un patrón que hace matching con una colección de elementos. El tipo de la colección con la que hace matching una plantilla, viene dada por el tipo de la colección referida. Una collection template expression puede definirse de tres maneras diferentes: enumeración, comprensión y selección de miembro. La interpretación del modelo en los tres casos es la siguiente:

- Enumeración. El conjunto de expresiones «parte» especifica exactamente los

elementos que comprenden la colección.

- Comprensión. La expresión obtenida del matching, que puede ser tanto una variable como una plantilla de objeto, que toma como valor un elemento de la colección. Si se usa una plantilla de objeto, entonces cada elemento de la colección debe hacer matching con el patrón especificado por una object template expression. Cada elemento en la colección debe satisfacer adicionalmente la expresión «parte».
- Selección de miembro. La expresión obtenida del matching solo puede ser una variable cuyo valor es un elemento de la colección. Si el tipo de la colección es una secuencia o un conjunto ordenado, la expresión toma como valor el primer elemento de la secuencia o del conjunto ordenado.

PropertyTemplateItem Los property template items se utilizan para especificar restricciones en los valores de las ranuras del elemento modelo que empareja el contenedor de la object template expression. La restricción está en la ranura, que es una instancia de la propiedad referida y de la expresión que contiene la restricción, y viene dada por el valor de la expresión.

Paquete QVTRelation

Relation Una relación o en inglés relation, es la unidad básica de especificación de comportamiento de una transformación en el lenguaje Relations. Ésta es una subclase concreta de «Rule». Una relation especifica la relación que debe cumplirse entre los elementos modelo de un conjunto de modelos candidatos, que conforma a los typed models de la transformación que contiene la relation. Una relación está definida por dos o más relation domains, que especifican los elementos modelo que han de estar relacionados, una cláusula when que especifica las condiciones necesarias bajo las cuales la relación debe cumplirse, y una cláusula where que especifica la condición que debe satisfacerse por los elementos modelo que están siendo relacionados.

RelationDomain La clase «RelationDomain» especifica los dominios de una relación. Es una subclase concreta de «Domain». Un dominio de relación o en inglés relation domain, tiene una variable tipada llamada root variable, que puede hacer matching en un modelo de un tipo de modelo dado. Una relación de dominio especifica un conjunto de elementos modelo de interés en términos de un patrón de dominio (domain pattern), que puede ser visto como un grafo de nodos de objetos, con sus propiedades y links de asociación, y con un distinguido nodo raíz que es asociado a la variable raíz de la relación de dominio.

DomainPattern La clase «DomainPattern» es una subclase de la clase «Pattern». Un patrón de dominio o en inglés domain pattern, puede especificar un grafo arbitrariamente complejo en términos de una template expression basada en object template expressions, collection template expressions y property template items. Un domain

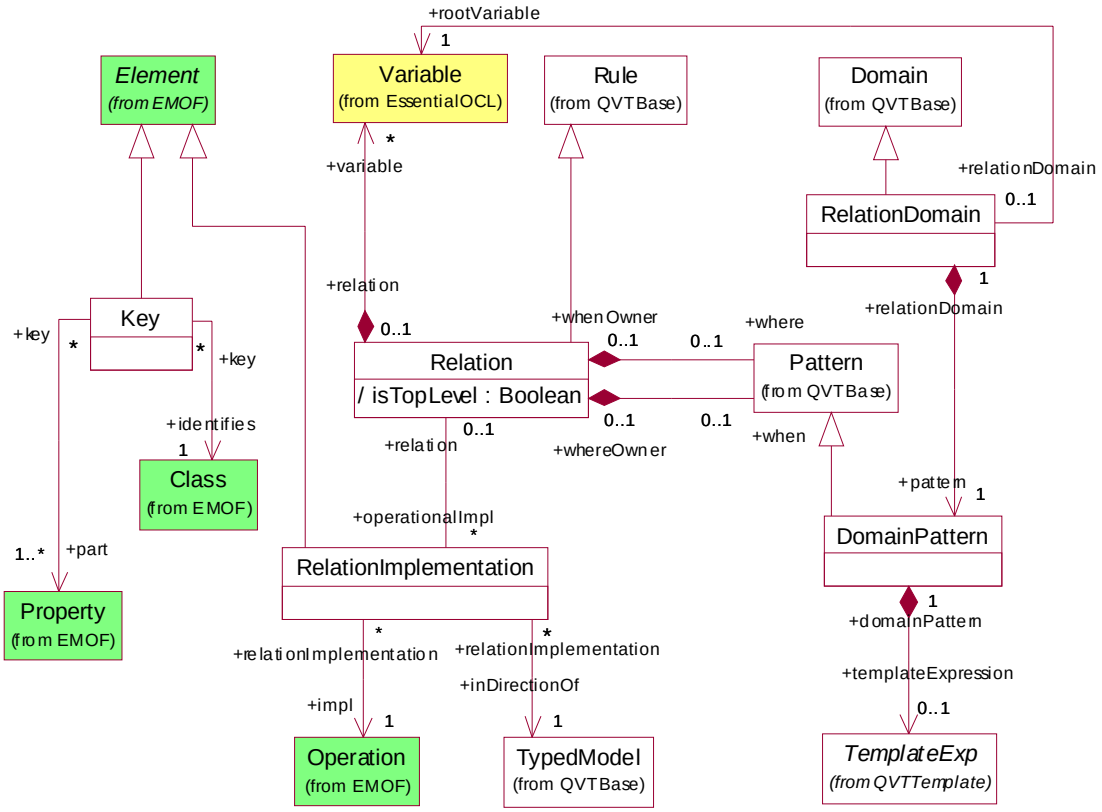


Figura 3.6: Paquete QVTRelation.

pattern tiene una distinguida template expression raíz, que es requerida para ser asociada a la variable raíz de la relación de dominio que contiene el patrón de dominio. Un object template expression puede tener otras template expressions anidadas con una profundidad arbitraria.

Key Una key define un conjunto de propiedades de una clase que identifica de manera única una instancia de clase en un modelo. Una clase puede tener múltiples keys.

RelationImplementantion Una RelationImplementation, especifica de manera opcional una implementación operacional de black-box para forzar un dominio de una relación. La operación black-box se invoca cuando la relación se ejecuta en la dirección del tipo de modelo asociado con el dominio forzado, y la relación se evalúa a falso de acuerdo con la comprobación semántica. La operación invocada es responsable de hacer los cambios pertinentes en el modelo para satisfacer la relación especificada. La signatura de la operación puede derivarse de la especificación del dominio de la relación como un parámetro de salida, correspondiente al dominio forzado, y como un parámetro de entrada, correspondiente a cada uno de los otros dominios.

Capítulo 4

Soporte tecnológico a la Ingeniería dirigida por modelos

4.1. Eclipse

Eclipse es un proyecto de desarrollo software de código abierto, cuyo propósito es proporcionar una plataforma de herramientas altamente integradas. El trabajo en Eclipse consiste en un proyecto central que incluye un framework genérico para la integración de herramientas, y un entorno de desarrollo Java construido usando el framework anterior. Otros proyectos extienden el framework núcleo para soportar tipos de herramientas y entornos de desarrollo específicos, entre los que encontramos EMF. Los proyectos en Eclipse se implementan en Java y se ejecutan en diversos sistemas operativos, incluyendo Windows y Linux.

Eclipse.org es un consorcio de diversas compañías que se han comprometido en proporcionar soporte al proyecto Eclipse en términos de tiempo, experiencia, tecnología o conocimiento. Los proyectos que conforman Eclipse operan bajo un organigrama bien definido que marca los roles y responsabilidades de los diversos participantes, incluyendo el consejo, los usuarios de Eclipse, los desarrolladores y los comités de gestión de proyectos.

La plataforma Eclipse es un framework para contruir IDEs. Se describe como «un entorno de desarrollo integrado para todo y nada en particular» [51]. Simplemente define la estructura básica de un IDE. Herramientas específicas extienden este framework, y se «enchufan» en él para definir un IDE particular colectivamente.

La unidad básica de función, o un componente, se denomina plug-in en Eclipse. La plataforma Eclipse misma, y las herramientas que la extienden se componen de plug-ins. Una sola herramienta puede consistir en un único plug-in, pero herramientas más complejas se dividen típicamente en varios.

Desde una perspectiva de empaquetado, un plug-in incluye todo lo necesario para ejecutar un componente, como código Java, imágenes, texto traducido, etc. También incluye

un archivo de manifiesto, llamado «plugin.xml», que declara las interconexiones con otros plug-ins. Indica, entre otras cosas, las siguientes:

- Requiere (Requires)– sus dependencias con otros plug-ins.
- Exporta (Exports)– la visivilidad de sus clases públicas a otros plug-ins.
- Puntos de extensión (Extensión points)– declaraciones de funcionalidad que hace disponibles a otros plug-ins.
- Extensiones (Extensions)– su uso de los puntos de extensión de otros plug-ins.

Al arrancar, la plataforma Eclipse descubre todos los plug-ins disponibles y casa las extensiones con sus correspondientes puntos de extensión. A continuación describiremos los principales *frameworks* del proyecto Eclipse directamente relacionados con sus capacidades de modelado *Eclipse Modeling Framework* (EMF) y *Graphical Modeling Framework* (GMF).

4.1.1. Eclipse Modeling Framework

Eclipse Modeling Framework es un framework de modelado para Eclipse. Este framework de modelado y generación de código permite definir un modelo de tres formas diferentes mediante Java anotado, XML Schema, o UML. Un modelo EMF es la representación de alto nivel común que une a las tres.

Un modelo EMF puede ser definido de cualquiera de estas tres maneras, siendo la potencia del framework y del generador la misma. A su vez, una vez definido un modelo EMF de cualquiera de estas formas, pueden obtenerse las otras de forma automática.

EMF es básica y simplemente un framework para describir un modelo y posteriormente poder generar otros elementos a partir de él. EMF es una tecnología que se mueve en la dirección de MDA, pero de forma lenta, ya que intenta integrar las ventajas del modelado pero desde el punto de vista del programador.

Un modelo EMF es esencialmente el subconjunto de los diagramas de clases de UML y podría considerarse como una implementación del lenguaje MOF propuesto por el OMG. Esto es, un simple modelo de las clases o datos de la aplicación. Por esto, un amplio porcentaje de los beneficios del modelado pueden obtenerse en un entorno de desarrollo Java estándar. La correspondencia estre un modelo EMF y el código Java que lo implementa es sencilla y natural.

4.1.1.1. Definiendo un modelo EMF.

No obstante a lo comentado anteriormente, un modelo se describe utilizando conceptos a un mayor nivel de abstracción que las meras clases y métodos. Notaríamos por

ejemplo, si observáramos la implementación de EMF, que los atributos corresponden a sendos métodos, para consultar y establecer sus valores. Igualmente, éstos, tienen la capacidad de notificar a los observadores (como una vista —View— de la interfaz, por ejemplo), o guardarse y recuperarse de un almacenamiento persistente. Las referencias son aún más potentes puesto que pueden ser bidireccionales, en cuyo caso la integridad referencial se mantiene. Las referencias pueden también persistirse entre diferentes recursos (documentos), donde entra en juego resolución delegada y la carga por demanda.

Para definir un modelo deberemos por tanto, disponer de una terminología común para describirlo. Y lo que es más importante, para implementar las herramientas de EMF y el generador, se requiere un modelo para la información.

El (Meta) modelo Ecore.

El modelo empleado para representar modelos en EMF se denomina Ecore. Ecore es también a su vez un modelo EMF, esto implica que Ecore es su propio metamodelo (o expresado en otras palabras, Ecore es un meta-metamodelo).

Gracias a Ecore, es posible definir los vocabularios locales de dominio que permiten el trabajo con modelos en distintos contextos.

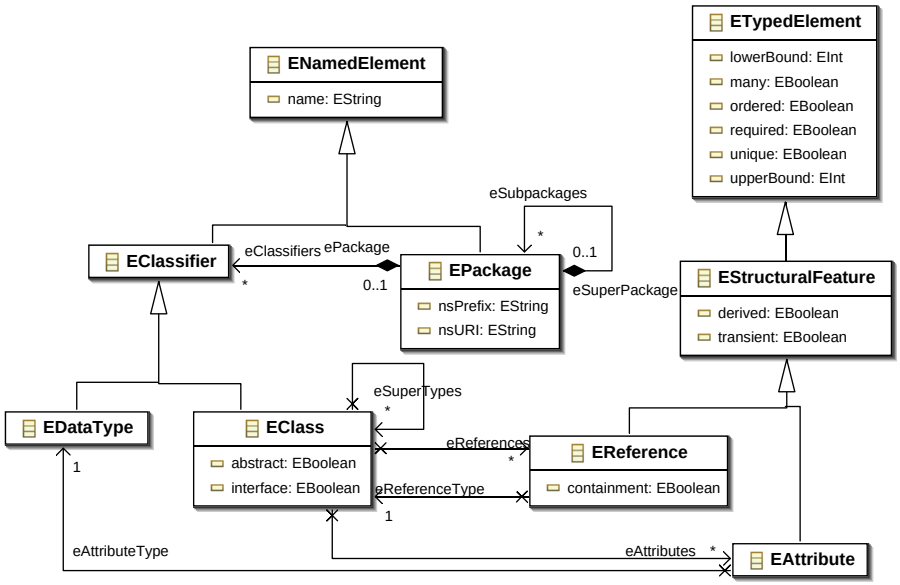


Figura 4.1: Subconjunto simplificado del modelo Ecore.

Ecore es un vocabulario diseñado para permitir la definición de cualquier tipo de metamodelos. Para ello, proporciona elementos útiles para describir conceptos y las relaciones entre ellos. En la figura 4.1 se muestra un subconjuto simplificado este modelo.

El elemento más importante es *EClass*, que modela el concepto de clase, con una se-

mántica similar al elemento Clase de UML. *EClass* es el mecanismo principal para describir conceptos mediante Ecore. Una *EClass* está compuesta por un conjunto de atributos y referencias, así como por un número de superclases (el símil con UML sigue siendo aplicable). A continuación se comentan el resto de los elementos aparecidos en el diagrama:

- *EClassifier*. Tipo abstracto que agrupa a todos los elementos que describen conceptos
- *EDataType* se utiliza para representar el tipo de un atributo. Un tipo de datos puede ser un tipo básico como int o float o un objeto, como por ejemplo java.util.Date.
- *EAttribute*. Tipo que permite definir los atributos de una clase. Éstos tienen nombre y tipo. Como especialización de ETypedElement, EAttribute hereda un conjunto de propiedades como cardinalidad (lowerBound, upperBound), si es un atributo requerido o no, si es derivado, etc.
- *EReference*. Permite modelar las relaciones entre clases. En concreto EReference permite modelar las relaciones de asociación, agregación y composición que aparecen en UML. Al igual que EAttribute, es una especialización de *ETypedElement*, y hereda las mismas propiedades.

  Además define la propiedad containment mediante la cual se modelan las agregaciones disjuntas (denominadas composiciones en UML).
- *EPackage* agrupa un conjunto de clases en forma de módulo, de forma similar a un paquete en UML. Sus atributos más importantes son el nombre, el prefijo y la URI. La URI es un identificador único gracias al cual el paquete puede ser identificado unívocamente.

Las similitudes de EMF con UML son evidentes, y es que EMF es un subconjunto de MOF, el cual a su vez está basado en los elementos del diagrama de clases de UML.

La cuestión de porqué no se ha utilizado UML como lenguaje de modelado es sencilla: Ecore es un subconjunto pequeño y simplificado de UML. UML soporta un modelado mucho más ambicioso que el soporte básico que se proporciona en EMF. Por ejemplo, UML permite modelar el comportamiento de una aplicación, a parte de su estructura de clases.

En el contexto de EMF, un metamodelo está constituido por las clases contenidas en un *EPackage*. Sólo se considera este caso simple; otros casos, como por ejemplo un metamodelo compuesto por más de un *EPackage*, no han sido tenidos en cuenta, sin que esto conlleve pérdida de genericidad o aplicabilidad.

Finalmente, para evitar confusiones cabe mencionar que en la documentación de EMF se utiliza la expresión modelo core para designar metamodelos. Dicha expresión hace referencia a modelos Ecore, es decir, modelos definidos utilizando el metamodelo Ecore. En cualquier caso el significado es el mismo: un metamodelo está constituido por un EPackage y un conjunto de *EClassifiers*

La creación de un modelo.

Ahora que ya disponemos de estos objetos Ecore para representar un modelo en memoria, el framework EMF puede leer de ellos para, entre otras cosas, generar código de implementación. La principal cuestión ahora es, ¿Cómo se crea un modelo Ecore?

Si se comienza mediante interfaces Java, el generador de EMF introspeccionará el código y construirá el modelo core. Si por el contrario se comienza a partir de un esquema XML el modelo se construirá a partir de éste. En caso de que se comience con UML, existen 3 posibilidades:

1. Edición directa en Ecore. Se puede editar un modelo en Ecore directamente, por ejemplo, por medio del editor en árbol de ejemplo de EMF, o mediante Omondo.

2. Importar desde UML. El asistente de nuevo proyecto EMF proporciona esta opción para archivos de Racional Rose (archivos .mdl) únicamente. Esto se debe a que fue la herramienta con la que se inició la implementación del mismo EMF.

3. Exportar desde UML. Básicamente es la misma opción que la anterior, salvo que la conversión se invoca desde la herramienta UML en lugar del asistente de nuevo proyecto EMF.

Serialización en XMI.

Hemos comentado que un modelo «conceptual» puede ser representado físicamente de al menos tres formas diferentes: código Java, XML Schema, o un diagrama UML. Pero de hecho, aún existe una cuarta forma de persistir un modelo que es la que se utiliza como representación canónica: XMI (XML Metadata Interchange).

La razón de utilizar XMI se debe a que es un estándar para serializar metadatos, lo cual es Ecore. Además, salvo el código Java, el resto de formas son opcionales. Si se utilizara Java para representar un modelo se debería inspeccionar el conjunto de archivos Java cada vez que se deseara representarlo.

Por esto, XMI es la elección más razonable para la forma canónica de Ecore. Es de hecho la forma más cercana a la tercera forma de representarlo (UML). El problema radica en que cada herramienta de UML tiene su propio formato de persistencia. Un archivo XMI de Ecore es una serialización estándar XML de los metadatos que EMF utiliza.

4.1.1.2. Generación de código.

La principal ventaja de EMF, como la del modelado en general es el aumento en la productividad que resulta de la generación automática de código. Dado un modelo Ecore que hemos definido, es posible obtener una implementación con unos pocos clicks. Todo lo que hay que hacer es crear un proyecto usando el Asistente para un nuevo proyecto EMF,

que automáticamente lanza el generador y seleccionar Generar código del modelo desde un menú.

4.1.2. Graphical Modeling Framework

GMF surge de la necesidad por parte de los desarrolladores de tener que usar frecuentemente los frameworks EMF y GEF de Eclipse para el desarrollo de sus herramientas basadas en modelos. GEF (Graphical Editing Framework) es el framework de Eclipse que permite a los desarrolladores crear editores gráficos *ricos* a partir de un modelo de aplicación. GEF está formado por dos plugins. El plugin *org.eclipse.draw2d* proporciona las herramientas para renderizar y ordenador los elementos al mostrar los gráficos. El framework GEF emplea una arquitectura de «modelo—vista—controlador».

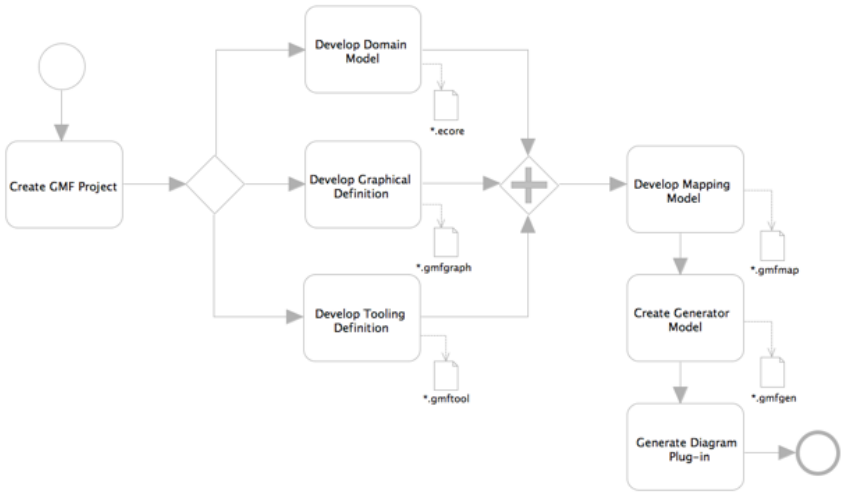


Figura 4.2: Proceso de creación de un editor gráfico mediante GMF.

La figura 4.2 ilustra los principales componentes y modelos usados durante un desarrollo basado en GMF. En la parte central de la figura se observa que un proyecto GMF parte de 3 modelos: *Domain Model*, *graphical Definition* y *Tooling Definition*. El primero de ellos, se corresponde con el modelo EMF para el que deseamos crear el nuevo editor gráfico. El segundo describe cuáles serán las primtivas gráficas que se dibujarán en el entorno de ejecución basado en GEF pero sin definicr ningún tipo de correspondencia con los elementos del modelo dominio para los que van a proporcionar capacidades de representación y edición. El tercer modelo permite definir las herramientas que se mostrarán en la paleta de dibujo del editor así como otros elementos de la interfaz gráfica (menús, barras de herramientas, etc.).

Generalmente, una definición gráfica puede ser igualmente válida para diferentes dominios. Por ejemplo, en el diagrama de clases de UML encontramos diferentes elementos

que son extremadamente parecidos en su apariencia y estructura. Un objetivo de GMF es que una definicón gráfica pueda ser reutilizada por distintos dominios. Esto se consigue mediante un modelo separado llamado *Mapping Model* que permite enlazar los elementos gráficos y las definiciones de herramienta con los elementos deseados del modelo dominio.

Una vez se han definido los enlaces apropiados, GMF proporciona un modelo generador para permitir afinar los últimos detalles de implementación para la fase de generación automática de código. La obtención del plugin de un editor basado en un modelo generador obtendrá un último modelo, llamado modelo *notacional*. El entorno de ejecución de GMF es el que enlaza este modelo notacional con el modelo dominio cuando el usuario está trabajando con un diagrama. A su vez, éste también proporciona las capacidades de persistencia y sincronización para ambos.

4.2. MOMENT. Un framework para la Gestión de Modelos.

MOMENT [7] es una herramienta que da soporte a los estándares propuestos por el OMG para dar soporte a transformaciones. La herramienta proporciona un soporte algebraico para las tareas de transformación y consulta de modelos mediante un eficiente sistema de reescritura de términos —Maude— y desde un entorno de modelado industrial —Eclipse Modeling Framework (EMF)—. Respecto a Maude, MOMENT aprovecha las capacidades de modularidad y parametrización de este sistema para proporcionar un entorno de transformación y consulta de modelos de forma genérica e independiente de metamodelo.

4.2.1. Espacios Tecnológicos y puentes.

Como se ha observado, MOMENT se desarrolla en dos ámbitos tecnológicos completamente diferenciados. A un lado, se encuentra la parte de interfaz del usuario y modelado, implementada como un nuevo framework para Eclipse; y al otro lado, el motor de cálculo: el álgebra de MOMENT ejecutándose sobre Maude. A cada uno de estos ámbitos diferenciados se le denomina «espacio tecnológico».

El concepto de espacios tecnológicos fue introducido en [40] en la discusión sobre el enlace de tecnologías heterogéneas. Un espacio tecnológico (ET) es un contexto de trabajo en el que se dispone de un conjunto de conceptos bien definidos, una base de conocimiento, herramientas, y una serie de posibilidades de aplicación específicas [34]. Un espacio tecnológico además suele ir asociado a una comunidad de usuarios/investigadores bien reconocida, un soporte educacional, una literatura común, terminología y saber hacer. Ejemplos de espacios tecnológicos son el ET XML, el ET DBMS, el ET de las sintaxis abstractas, el ET de las ontologías, el ET de MOF/MDA, en el que se enmarca UML, y el ET de EMF, que guarda un gran parecido con el anterior.

4.2.1.1. Puentes tecnológicos.

Cada espacio tecnológico tiene unas características que le hacen especialmente apropiado para resolver un tipo de problemas. Muchas veces sin embargo lo más apropiado es trabajar con varios ETs a la vez. Para ello existen o es posible definir enlaces o puentes entre espacios. Por ejemplo son bien conocidos los puentes de MDA al ET de sintaxis abstractas, o de UML al ET XML a través de XMI.

Un puente entre espacios puede ser bidireccional, como en los ejemplos comentados, o unidireccional, cuando no es posible reconstruir el artefacto origen.

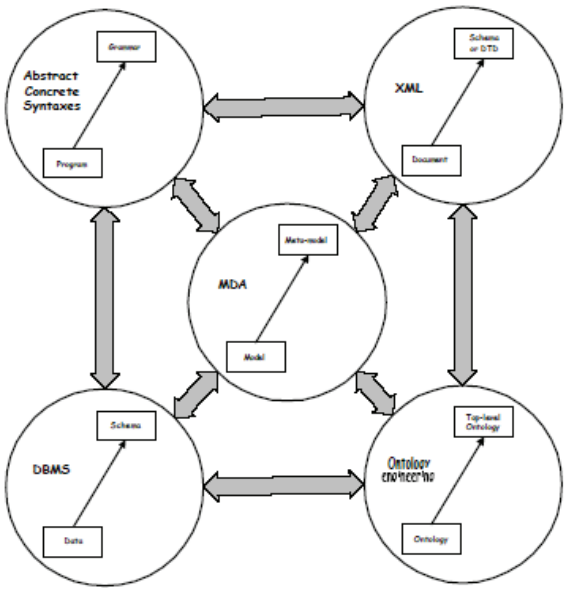


Figura 4.3: Cinco espacios tecnológicos y diversos puentes entre ellos [40].

En MOMENT los operadores de gestión de modelos [4] han sido especificados algebráicamente utilizando el formalismo Maude como se ha comentado anteriormente. El ET de Maude se caracteriza por las ventajas que aporta el formalismo de especificaciones algebraicas: abstracción, subtipado, modularización, genericidad mediante parametrización, etc.

Este ET también puede ser visto como un paradigma de modelado, considerando el álgebra universal de Maude como el lenguaje de definición de metamodelos en el nivel M3. En el nivel M2, los metamodelos son los módulos que proporcionan especificaciones algebraicas Maude.

MOMENT representa un modelo como una estructura de términos algebraicos, caracterizados por una especificación algebraica que proviene del metamodelo.

Para poder utilizar MOMENT desde la ingeniería de modelos será necesario dis-

poner de unos puentes tecnológicos entre ambos espacios tecnológicos. Este problema es resuelto en [32]. En dicho proyecto se resuelve la definición y construcción de estos puentes tecnológicos entre el ET de Maude y el ET de EMF.

Estos puentes creados permiten representar un modelo como un término algebraico, manipularlo desde Maude, y devolverlo como un modelo EMF. Se ha escogido EMF dentro del campo MDE por su interoperabilidad.

Se espera que EMF sea una puerta de entrada a otros entornos MDE, y que de esta manera el trabajo realizado sirva para habilitar de manera lo más completa posible la interoperabilidad de MOMENT con MDE.

4.2.2. Visión global del framework MOMENT.

En MOMENT los modelos se especifican como conjuntos de elementos de forma independiente del metamodelo, de manera que los operadores pueden acceder a los elementos sin conocer la representación de un modelo. La interfaz de MOMENT está integrada en EMF, de manera que el formalismo de especificaciones algebraicas queda totalmente transparente al usuario.

Para ilustrar el funcionamiento de MOMENT, se indica un pequeño ejemplo de integración de esquemas XML. Para ello se ha definido una parte del metamodelo del lenguaje de definición XML (XSD), mostrado en notación UML en la figura 4.4.

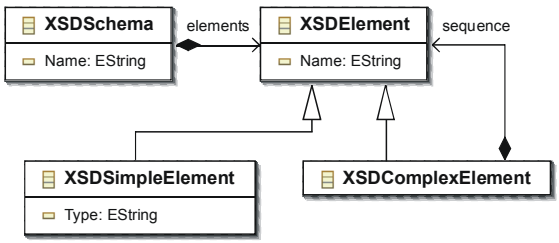


Figura 4.4: Parte del metamodelo XSD.

Utilizando el editor en forma de árbol que proporciona EMF, definimos los esquemas XML A y B en la 4.5. Se aplica el operador Merge a ambos, obteniendo el esquema XML integrado C y dos modelos de trazas (mapAC y mapBC) que enlazan los elementos de los modelos de entrada con los elementos del modelo de salida. La invocación del operador es la siguiente: $< C, mapAC, mapBC >= Merge(A, B)$.

Para poder realizar la integración de los esquemas XML, estos deben ser traducidos a términos en Maude, para que el operador Merge pueda aplicarse sobre ellos.

Entre las numerosas herramientas que dan soporte a la ingeniería de Modelos, MOMENT utilizará EMF como entorno de modelado para nuestra herramienta de Gestión de Modelos por su situación dentro del marco de la Ingeniería de Modelos. EMF permite tratar

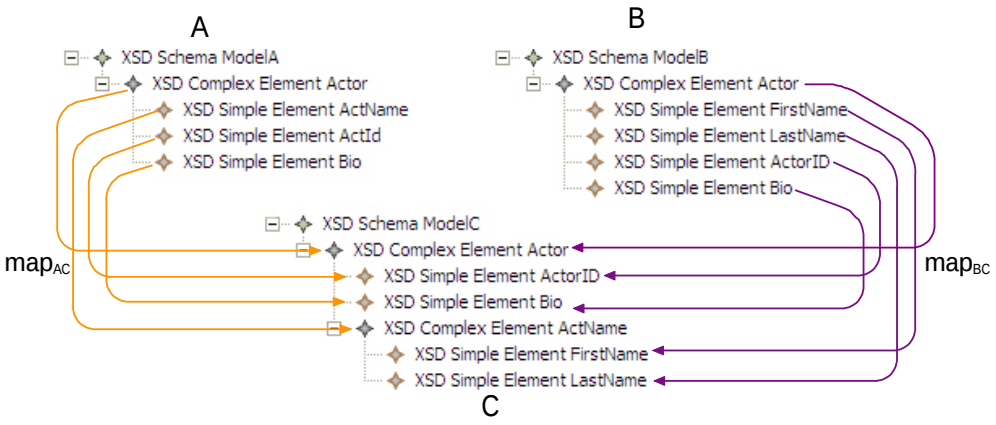


Figura 4.5: Aplicación del operador Merge.

con gran variedad de artefactos software, como esquemas XML, modelos UML (definidos en entornos visuales de modelado como Rational Rose), esquemas relacionales (a través de Rational Rose), ontologías, entre otros. Además, EMF es utilizada por las principales herramientas de IBM, aportando una visión industrial a nuestro enfoque de Gestión de Modelos.

4.2.3. Marco conceptual para la representación de artefactos software en Maude.

Siguiendo un enfoque de Ingeniería de Modelos, para tratar con artefactos software utilizamos la terminología que define el estándar Meta-Object Facility de la iniciativa MDA. Este estándar, como se comentó en el apartado 3.1.2, presenta una arquitectura de cuatro capas de modelado que permite clasificar artefactos software con diferente propósito: M3 (metametamodelos), M2 (metamodelo), M1 (modelo), M0 (sistema real).

Una estrategia para trabajar con metamodelos consiste en definir una sintaxis básica en el nivel M3, que pueda ser utilizada para definir artefactos software en niveles inferiores. En EMF, el metametamodelo se llama Ecore y proporciona una serie de primitivas de modelado: un subconjunto del diagrama de clases del metamodelo UML. Estas primitivas se utilizan para definir metamodelos en el nivel M2, constituyendo un paradigma de modelado. Como por ejemplo, el lenguaje de definición de esquemas XML (XML Schema Definition language — XSD).

Los elementos de un metamodelo son utilizados como tipos para definir los elementos que constituyen un modelo en el nivel M1. En el caso del metamodelo XSD, un modelo es un esquema XML específico. Los elementos de un modelo también se comportan como tipo para definir información en el nivel M0 de la arquitectura MOF. Por ejemplo, un esquema XML define los elementos que se pueden utilizar en un documento XML.

4.2.4. Proyecciones de artefactos software EMF sobre Maude.

Un espacio tecnológico se caracteriza por el soporte tecnológico que se proporciona a un determinado paradigma de modelado. Cada paradigma de modelado se organiza entorno a un metametamodelo común y persigue unos objetivos específicos.

El espacio tecnológico EMF se caracteriza por las facilidades que ofrece para representar una buena variedad de artefactos software como modelos y por su interoperabilidad con otras herramientas industriales de modelado. El espacio tecnológico Maude se caracteriza por las ventajas que aporta el formalismo de especificaciones algebraicas. Este ET también puede ser visto como un paradigma de modelado, considerando el lenguaje Maude como el lenguaje de definición de metamodelos en el nivel M3. En el nivel M2, los metamodelos son los módulos que proporcionan especificaciones algebraicas Maude.

Una especificación algebraica constituye la visión como instancia de un determinado metamodelo, proporcionando la descripción sintáctica de las primitivas (llamadas constructores en el campo de las especificaciones algebraicas), necesarias para especificar un artefacto software en el nivel M1. Cuando la especificación algebraica es interpretada como álgebra, se obtiene la visión como tipo del metamodelo, donde los constructores se pueden utilizan para definir artefactos software en el nivel M1. Éstos son representados sintácticamente como términos, representando la información en forma de árbol.

Para manipular modelos EMF con los operadores algebraicos de MOMENT se han definido una serie de proyecciones entre ambos espacios tecnológicos tal y como muestra la figura 4.6. Estas proyecciones permiten representar un modelo como un término algebraico, manipularlo desde Maude, y devolverlo como un modelo EMF.

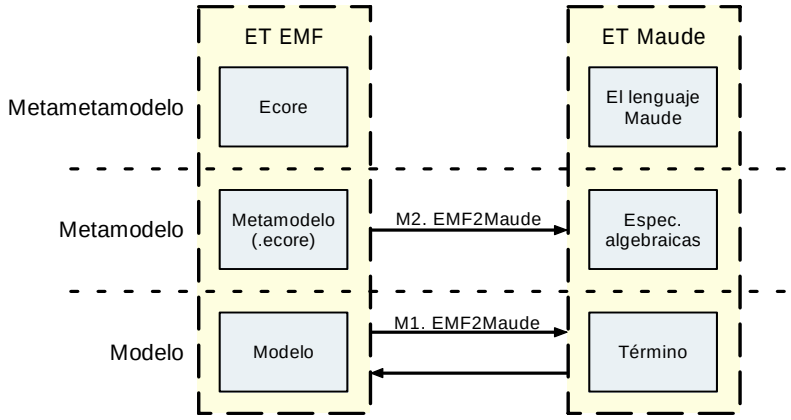


Figura 4.6: Enlaces entre el ET EMF y el ET Maude.

4.2.4.1. Interoperabilidad en el nivel M2

En el nivel de metamodelos se establece un enlace unidireccional que permite la proyección de un metamodelo EMF sobre el ET Maude, obteniendo una especificación algebraica. De este modo, un metamodelo se interpreta como un álgebra que proporciona los constructores necesarios para definir modelos y las operaciones necesarias para manipularlos, en el contexto de la Gestión de Modelos.

Este enlace es unidireccional pues los metamodelos se especifican mediante herramientas visuales de modelado a través de EMF (el nombre que hemos asignado a este enlace es M2-EMF2Maude). Este hecho permite hacer transparente el uso del formalismo Maude al usuario final del framework MOMENT.

MOMENT trabaja directamente sobre un álgebra de operadores genéricos de manipulación de modelos. Estos operadores pueden ser adaptados a un metamodelo específico haciendo uso de las capacidades de parametrización que ofrece Maude, basándose en el concepto formal de Pushout [15] de teoría de categorías.

En el diagrama del mecanismo de paso de parámetros (figura 4.7), TRIV constituye el parámetro formal del módulo parametrizado MOMENT-OP(X::TRIV). La especificación sigXSD constituye el parámetro actual para el módulo parametrizado. sigXSD proporciona los constructores correspondientes a las primitivas de un metamodelo específico. Esta especificación algebraica está relacionada con el parámetro formal mediante la vista vXSD. Como ejemplo se ha utilizado el metamodelo XSD simplificado. En él, a partir de la metainformación que describe la clase XSDSimpleElement, que indica como definir un elemento simple en un esquema XML, se obtiene un constructor que permite definir un elemento simple en un esquema XML como un término del álgebra.

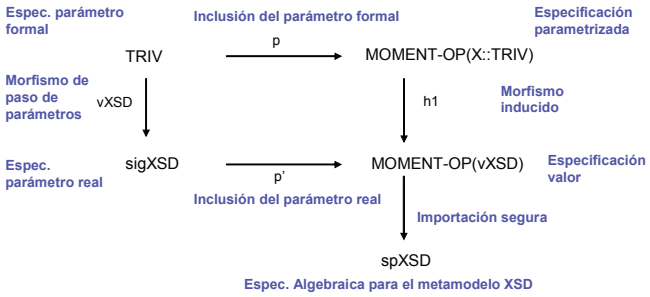


Figura 4.7: Diagrama del mecanismo de paso de parámetros en Maude.

El módulo sigXSD es importado por el módulo spXSD, en el que se extiende la presentación axiomática de los operadores genéricos adaptándolas a un metamodelo específico. Por ejemplo, el operador Merge, cuando se utiliza en el metamodelo XSD, permite integrar esquemas XML. Para definir este tipo de integración de forma más precisa, se pueden añadir relaciones de equivalencia que tengan en cuenta las primitivas del metamodelo XSD: elemento simple, elemento complejo, etc. Estas relaciones de equivalencia específicas

al metamodelo XSD se añaden al módulo spXSD en forma de axiomas. La especificación algebraica resultante contituye el álgebra de un metamodelo. El módulo spXSD también es generado automáticamente por la herramienta MOMENT a partir de la especificación de los nuevos axiomas en interfaces visuales.

4.2.4.2. Interoperabilidad en el nivel M1

Existe otro tipo de enlace entre el ET EMF y el ET Maude en el nivel de modelos. Este enlace es bidireccional y consta de dos tipos de proyecciones:

- M1-EMF2Maude: Este mecanismo proyecta un modelo EMF, definido mediante un metamodelo EMF, sobre el ET Maude como un término. Para proyectar un modelo sobre el ET Maude, la herramienta MOMENT consulta el correspondiente metamodelo y obtiene el constructor de la correspondiente especificación algebraica, que es necesario para especificar el término de forma automática.

- M1-Maude2EMF: Este mecanismo proporciona la proyección inversa a la anterior, obteniendo un modelo EMF a partir de un término Maude. En este paso, cuando la herramienta MOMENT lee un término que representa un modelo, determina las primitivas del metamodelo EMF que debe utilizar para construir el modelo EMF a partir de los símbolos de los constructores utilizados en el término. Con estas primitivas se construye el modelo dinámicamente y se persiste en formato XMI.

Los enlaces, que han sido descritos entre el ET EMF y el ET Maude, permiten la aplicación de operadores algebraicos sobre modelos definidos de forma gráfica mediante entornos industriales de modelado. Por ejemplo, supongamos que se desea realizar la integración de dos esquemas XML, cuyo metamodelo ha sido definido mediante Ecore. El proceso seguido es el siguiente:

1. Se obtiene la especificación algebraica spXSD a partir del metamodelo XSD.

2. Se proyectan los esquemas XML A y B a términos del álgebra, interpretada a partir de spXSD.

3. Se aplica el operador Merge a ambos términos y se obtiene el término resultante mediante el mecanismo de reducción de Maude. Como resultado de la operación de integración se obtiene el término C[1], que representa el esquema XML integrado.

4. Finalmente, el término C es proyectado al ET EMF como un modelo.

[1]El operador Merge también produce dos modelos de trazabilidad que relacionan los modelos de entrada A y B con el modelo de salida C, respectivamente. Ambos modelos se han obviado para simplificar el ejemplo.

4.2.5. Presentación del álgebra de operadores de Gestión de Modelos de MOMENT.

Puesto que las correspondencias entre modelos se definen de forma implícita, es posible aplicar un operador genérico a dos modelos cualquiera. De la misma manera, permite obtener de forma automática el modelo de correspondencias entre ambos modelos.

La forma de funcionamiento, es la siguiente: en MOMENT, los operadores están definidos en un módulo parametrizado llamado MOMENT-OP. De esta forma, los operadores están definidos de forma genérica. Para aplicar estos operados a modelos específicos, este módulo debe ser instanciado pasando un metamodelo como parámetro actual. Esta tarea la realiza automáticamente la herramienta MOMENT.

A continuación, mostramos algunos ejemplos de operadores de Gestión de Modelos indicando sus entradas, salidas y semántica.

4.2.5.1. Operadores comunes

1. *Cross* y *Merge*: Estos operadores corresponden a operaciones de conjuntos bien definidas: intersección y unión disjunta respectivamente. Ambos operadores reciben dos modelos (A y B) como entradas, y producen un tercer modelo (C). El operador *Cross* devuelve un modelo C que contiene elementos que participan en ambos modelos de entrada (A y B); mientras que el operador *Merge* devuelve un modelo C que contiene los elementos que pertenecen tanto al modelo de entrada A como a B, eliminando los elementos duplicados. Ambos operadores también devuelven a su vez sendos modelos de enlaces (map_{AC} y map_{BC}) que relacionan los elementos de cada modelo de entrada con los elementos del modelo resultante. Por ejemplo: $< C, map_{AC}, map_{BC} >= Cross(A, B)$.

2. *Diff*. Este operador realiza la diferencia entre dos modelos de entrada (A y B). La diferencia entre dos modelos (C) es el conjunto de elementos del modelo A que no corresponden a ningún elemento del modelo B. Este operador devuelve un único modelo de enlaces (map_{AC}). El modelo de mappings map_{BC} es innecesario puesto que se sabe a priori que será vacío.

3. *ModelGen*. *ModelGen* realiza la traslación de un modelo A, que conforma a un metamodelo origen MMA, a un metamodelo MMB destino, obteniendo el modelo B. Esta transformación implica tratar con dos metamodelos. Esto es perfectamente factible en nuestra aproximación, dada la modularidad y reusabilidad que las especificaciones algebraicas proporcionan. Este operador produce a su vez un modelo de correspondencias (map_{AB}) relacionando los elementos del modelo de entrada con los elementos del modelo generado. Por ejemplo: $< B, map_{AB} >= ModelGenMMA2MMB(A)$.

Como se observa en los ejemplos mencionados, toda aplicación de un operador sobre uno o varios modelos de entrada para producir los correspondientes modelos de salida, pro-

ducen de forma automática los modelos de correspondencias que relacionan las entradas (modelos dominio) con las salidas (modelos rango). indicando que ambos lados de cada enlace representan el mismo elementos en modelos diferentes. Diversas aproximaciones, como por ejemplo RONDO, especifican operadores basados en estas correspondencias para tratar con modelos. Esto implica que las correspondencias entre ambos modelos, deben definirse de forma explícita para aplicarse sobre los modelos [44]. Sin embargo, en MOMENT, esta relación se establece de forma implícita mediante un morfismo de equivalencia definido a nivel de metamodelo (nivel M2), y no de modelo (nivel M1), de forma más abstracta y reusable.

4.2.5.2. Operadores de soporte a la navegación

Los operadores que proporcionan soporte para la navegación lo hacen a través de un modelo de trazabilidad con los siguiente elementos: dos modelos de entrada (A y B); un modelo de trazabilidad (map_{AB}) que relaciona los elementos de dos modelos de entrada y que ha sido generado automáticamente por un operador, o manualmente por un usuario; un modelo (A') que es un submodelo de A (esto es, que A' solo contiene elementos que pertenecen a A); y un modelo (B') que es un submodelo de B. Los operadores de trazabilidad considerados aquí son:

1. *Domain* y *Range*. Estos operadores proporciones la navegación hacia delante y hacia atrás a través de un modelo de trazabilidad, respectivamente. Ambos operadores obtienen un modelo como resultado que no es un modelo de trazabilidad.

   El operador *Domain* toma tres modelos como entrada: un modelo de trazabilidad (map_{AB}), un modelo dominio (A), y un modelo rango (B'). El operador navega los enlaces del modelo de trazabilidad y devuelve un submodelo de A (A'), como se muestra en la figura 4.8.a.

   El operador *Range* también recibe tres entradas: un modelo de trazabilidad (map_{AB}), un modelo dominio (A'), y un modelo rango (B). Este operador realiza la operación inversa a la anterior: navega el los enlaces de trazabilidad que tienen elementos de A' como elementos dominio y devuelve un submodelo del modelo rango B (B'), como se muestra en la figura 4.8.b.

2. *SelectMappingsByDomain* y *SelectMappingsByRange*. Estos operadores producen un modelo de trazabilidad como salida y permiten seleccionar parte de un modelo de trazabilidad.

   El operador *SelectMappingsByDomain* recibe dos modelos de entrada: un modelo dominio (A') y un modelo de trazabilidad (map_{AB}). El operador extrae los enlaces de trazabilidad del modelo map_{AB} que tienen elementos del modelo A' como elementos dominio y devuelve este submodelo. Los enlaces de trazabilidad que se añadir al modelo de trazabilidad de salida se muestran en la figura 4.8.c con una línea punteada.

   El operador *SelectMappingsByRange* recibe dos modelos de entrada: un modelo rango (B') y un modelo de trazabilidad (map_{AB}). En este caso, el operador extrae los

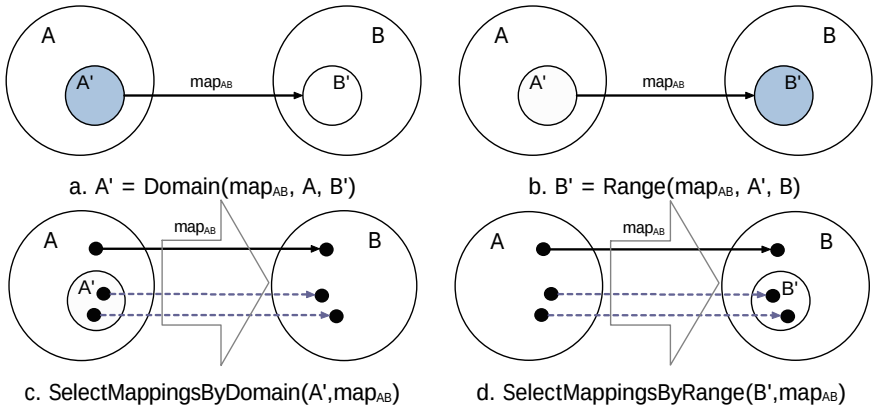


Figura 4.8: Operadores genéricos para navegación de las trazas.

enlaces de trazabilidad del modelo map_{AB} que tienen elementos de B' como elementos rango, y devuelve este submodelo, como se muestra en la figura 4.8.d.

4.2.6. Soporte para las transformaciones en MOMENT

En el proceso de transformación, el usuario interactúa con la plataforma MOMENT especificando un programa QVT utilizando el lenguaje Relations. MOMENT, proporciona un editor textual con coloreado de sintaxis que facilita la labor de especificar transformaciones.

Una vez especificado el programa QVT-Relations se inicia un proceso automático y totalmente oculto al usuario, que culmina con la ejecución en Maude de la transformación definida en el programa y la devolución al usuario de los resultados obtenidos. Este proceso se divide en tres partes fundamentales: análisis, proyección a Maude, y ejecución y proyección de los resultados a EMF (figura 4.9).

Para llevar a cabo estas tres fases, son necesarios un conjunto de componentes funcionales que forman la arquitectura de MOMENT QVT:

- QVT Parser: encargado de analizar un programa QVT Relations de entrada y generar su modelo correspondiente.
- MOMENT Registry: repositorio de artefactos generados y/o utilizados por MOMENT.
- MOMENT Relations: encargado de generar y proyectar código Maude a partir de la especificación de una transformación.
- OCL Parser: encargado de analizar expresiones OCL y generar su código Maude correspondiente.

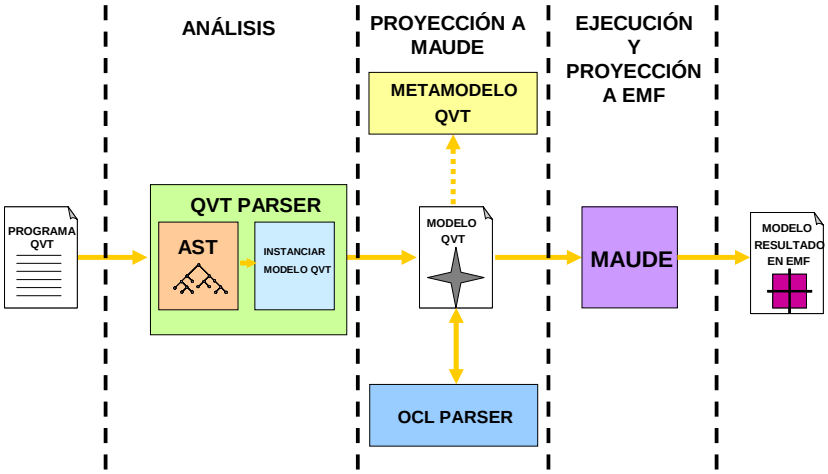


Figura 4.9: Proceso de ejecución de un programa QVT Relations en MOMENT.

4.2.6.1. Fase de análisis

Esta etapa comienza con un análisis léxico y sintáctico del programa QVT Relations. Para este análisis es necesario consultar los metamodelos respectivos a los modelos participantes en la transformación. Por consiguiente, es prerrequisito imprescindible que estos metamodelos estén registrados en EMF.

4.2.6.2. Ejecución y proyección a EMF.

Llegado este punto, se dispone en el espacio tecnológico Maude de toda la información necesaria para lanzar la transformación a ejecución. Desde Maude, se aprovecha el mecanismo de reducción modulo las ecuaciones generadas a partir de un programa QVT Relations, como brevemente se indica en el apartado anterior y de pattern-matching, obteniendo el término resultante que en el ejemplo anterior se corresponde con un esquema relacional. Finalmente, este término es proyectado de vuelta a EMF utilizando los puentes definidos a nivel M1 entre ambos espacios tecnológicos. Se pueden encontrar más detalles del proceso de compilación a código Maude en [57].

Igualmente, para conocer en mayor detalle el funcionamiento de la herramienta MOMENT-QVT desde el punto de vista del usuario, se puede consultar el el anexo 7, donde se encuentra un manual de usuario. Este manual ha surgido desde la experiencia adquirida en el uso de MOMENT-QVT para el casod e estudio que este trabajo presenta.

4.3. Medini QVT

4.3.1. Historia

MediniQVT es una herramienta enteramente implementada en Java, e integrada en Eclipse como un conjunto de plugins para dar soporte a transformaciones de modelos empleando el estándar QVT del OMG. Ha sido desarrollada por la empresa *ikv++ technologies ag* [33], con sede en Alemania. El motor fue inicialmente lanzado como un producto cerrado aunque gratuito en 2007, y fue a inicios del año 2008 cuando se liberó el código fuente bajo la licencia de código abierto EPL —*Eclipse Public License*—.

4.3.2. Características

Desde el punto de vista de la tecnología, MediniQVT se basa en EMF como entorno de modelado y metamodelado. Para la definición de transformaciones emplea el lenguaje QVT-Relations, dando soporte a transformaciones modelo-a-modelo, e internamente, el motor de evaluación de expresiones OCL está basado en OSLO (*Open Source Library for OCL [21]*).

Las principales características de la herramienta completa son:

- Ejecución de transformaciones QVT expresadas en la sintaxis concreta textual del lenguaje QVT-Relations
- Editor con resaltado y completado de código.
- Depurador para trazar la ejecución de transformaciones paso a paso a través de las reglas.
- Implementación del concepto de clave (*key*) del lenguaje QVT, permitiendo las transformaciones para actualizaciones incrementales.
- Transformaciones con n-dominios participantes.
- Transformaciones bidireccionales (cuando la definición de éstas así lo permite).

Toda esta funcionalidad está implementada en diversos plugins de Eclipse. En este sentido, debe aclararse que es únicamente el plugin que contiene el motor de ejecución de transformaciones el que es público y de código abierto. Todos los plugins que implementan funcionalidad adicional (editor textual, depuración, etc.) son de código cerrado.

Por ello, para la construcción del prototipo presentado en este trabajo, sólo se ha tomado aquella parte que es de código abierto y reutilizable, habiéndose implementado exprésamente para el trabajo un conjunto de plugins y herramientas que permiten la invocación de forma amigable de transformaciones QVT. En la sección 6.4 se describe en

detalle el conjunto de plugins desarrollados, y en la sección 8 se muestra indica su guía de uso.

Capítulo 5

Un enfoque de DSDM en la migración de datos biológicos.

En el trabajo inicial sobre el estudio del *pathway* del TLR4 la migración de datos desde la base de datos origen hasta la herramienta de simulación para su representación mediante una red de Petri coloreada debe realizarse de forma manual. A continuación se muestra una solución al problema de migración de datos del caso de estudio empleando transformaciones, según la filosofía de desarrollo de software dirigido por modelos. Esto supone las siguientes tareas:

1. Desarrollo del modelo de datos del dominio origen (TRANSPATH®).
2. Desarrollo del modelo de datos del dominio destino (*CPN Tools*).
3. Definición mediante un lenguaje de transformaciones las reglas de transformación entre el dominio origen y el dominio destino.
4. Implementación del mecanismo de preprocesado de datos permitiendo reconstruir los datos originales como instancias del modelo origen.
5. Definición del postprocesado de los datos, que implementa el mecanismo de persistencia al formato final.

A continuación se presenta la solución proporcionada. En primer lugar, se describe el proceso de transformación y sus etapas, en segundo lugar se describen los modelos del dominio origen y el modelo destino, y por último, se describe en mayor detalle el proceso de transformación.

Esta aproximación proporciona diversas ventajas frente a aproximaciones tradicionales. En primer lugar, permite automatizar un proceso que se hacía de forma manual. En segundo lugar, define una estructura modular para la aplicación encargada de la migración de los datos, donde el formato de persistencia de estos es independiente del proceso de

transformación. En tercer lugar, el uso de un lenguaje declarativo define las correspondencias entre el dominio origen y destino de forma más clara y precisa (en contraposición a una implementación imperativa). Por último, las capacidades de trazabilidad que se proporcionan de forma implícita permiten trazar desde la herramienta de simulación hacia la base de datos origen aquellos datos que se encuentre que son erróneos.

5.1. Arquitectura e implementación de la herramienta.

El proceso de migración de datos, tal y como se deriva de las tareas anteriores, se realiza en tres pasos: (1) recuperación y pre-procesado de los datos, (2) ejecución de la transformación mediante un motor de transformaciones y (3) post-procesado y persistencia de los datos. En una aproximación de DSDM, el uso de un motor de transformaciones implica que se deben definir en primer lugar los modelos origen y destino de la transformación para poder establecer las correspondencias entre uno y otro posteriormente.

La solución aquí presentada hace uso de MOMENT. MOMENT es una herramienta integrada en el entorno Eclipse que proporciona soporte para transformaciones. Como entorno de modelado, MOMENT emplea el *Eclipse Modeling Framework* (EMF). Este *framework* proporciona como lenguaje de modelado Ecore. Además, emplea como formato de persistencia XMI. EMF permite –entre otros— la creación de modelos Ecore a partir de un esquema XSD. En el caso de estudio, tanto los ficheros origen (datos extraídos de la base de datos de TRANSPATH®) como los ficheros destino (archivo de *CPN Tools*) son ficheros XML de los que se dispone de la definición de la estructura que validan (ya sea un esquema XSD —XML Schema Definition—, o su DTD —Document Type Definition— correspondiente).

No obstante, los modelos Ecore obtenidos a partir de esquemas XSD son complejos y no siempre representan la semántica de los datos de forma clara. La figura 5.1 muestra dos pequeños fragmentos de muestra del esquema XML de la base de datos TRANSPATH®representado en el editor en árbol de EMF. Este esquema XML se ha generado de forma automática a partir del DTD que se proporciona entre los recursos de TRANSPATH®, y que se adjunta en el anexo C.

Como se ha podido observar en la figura 5.1 existen numerosos elementos que hacen complejo tratar directamente con el esquema XML como si de un metamodelo Ecore cualquiera se tratase. Además, se pierde expresividad en los modelos recuperados. Obsérvese en la captura derecha, cómo los elementos chains, reactions_involved, reactions_involved (entre muchos otros ejemplos) únicamente se definen como secuencias de ítems genéricos, cuando por ejemplo chains contiene una lista chains (cadenas de reacciones), y reactions_involved y reactions_involved contienen reacciones. Por todo esto, se ha decidido definir de forma manual los modelos origen y destino, teniendo sólo en cuenta aquella información relevante para el caso de estudio.

La figura 5.2 muestra la arquitectura de la herramienta. En ella se reflejan de forma clara los 3 pasos necesarios para realizar la migración de los datos. En primer lugar se

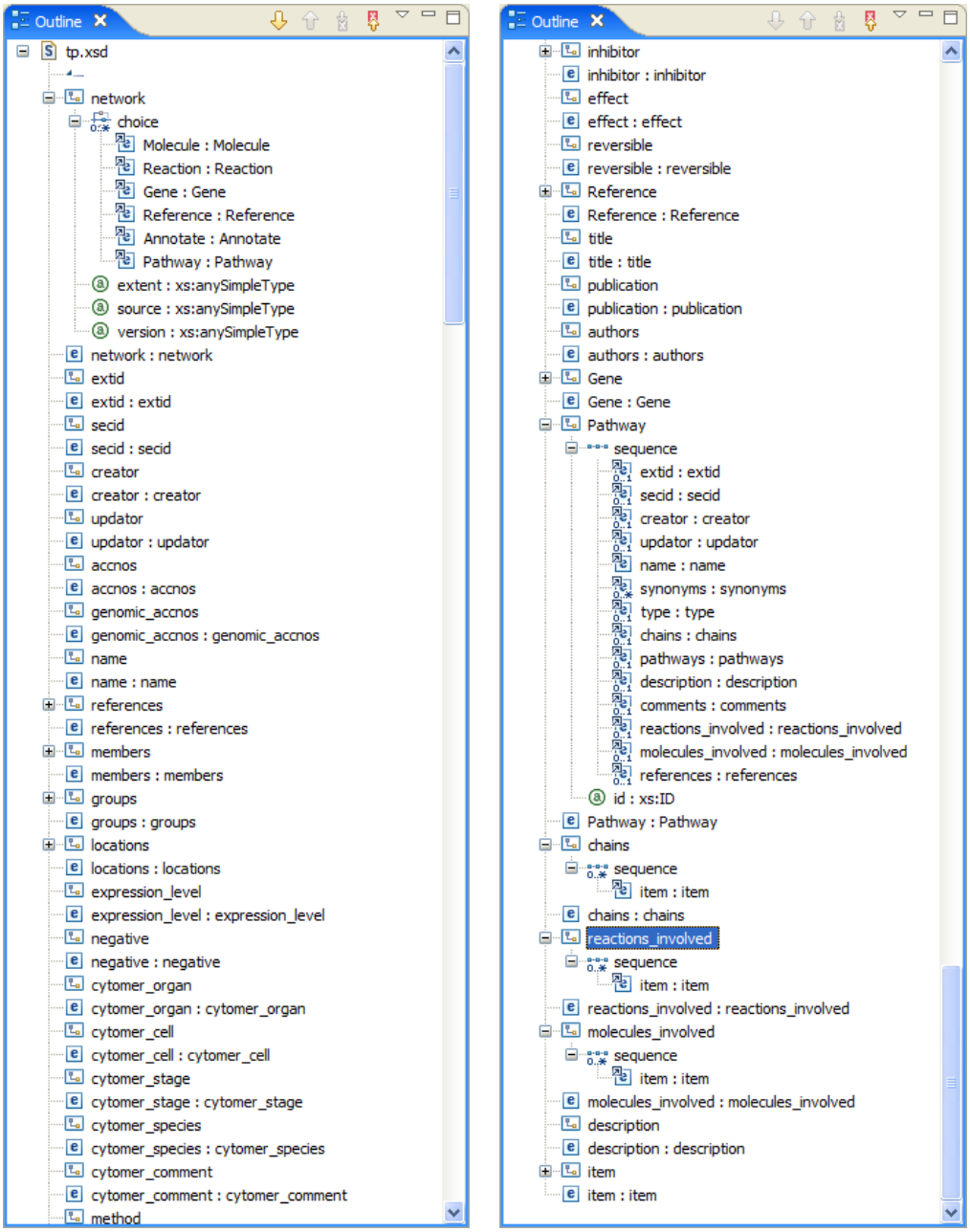


Figura 5.1: Ejemplo de esquema XML importado en Eclipse

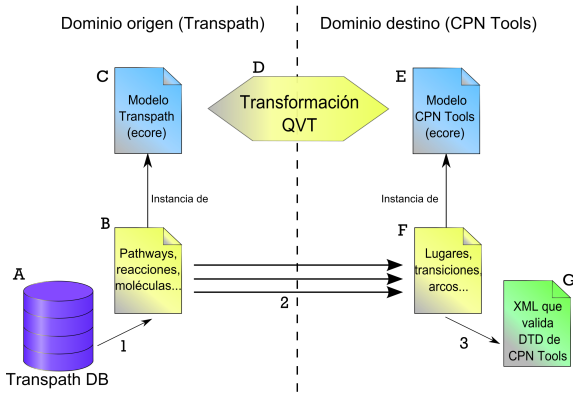


Figura 5.2: Arquitectura de la herramienta.

extraen los datos de la base de datos TRANSPATH®(A), y se reconstruyen como una instancia en XMI (B) del modelo Ecore de TRANSPATH®(C). Este primer paso (1) se realiza de forma sencilla en Java ya que la correspondencia de los datos origen con el modelo en EMF es directa. La implementación de este primer paso de pre-procesado de los datos es una adaptación del trabajo realizado en [76].

El segundo paso de la migración (2) es el principal y más complejo. Se realiza mediante MOMENT y su motor de transformaciones, ejecutando la transformación del dominio de TRANSPATH®(reacciones, moléculas, etc.) al dominio de *CPN Tools* (lugares, transiciones, arcos, etc.). Tras la definición de las reglas de transformación (D) entre los dominios origen (C) y destino (E), se aplica la transformación sobre los datos recuperados de la base de datos (B) obteniendo una instancia en el dominio de *CPN Tools* (F).

Por último, el tercer paso de la migración (3) es de nuevo un proceso trivial en el que se persisten los datos desde EMF (F) a un fichero XML comprensible por la herramienta *CPN Tools* (G). En este, paso además, pueden realizarse otras tareas de post-procesado, como por ejemplo, la inclusión de algún algoritmo de redibujado (*layout*) sobre los elementos de la red de Petri.

5.2. Modelos de Datos.

5.2.1. TRANSPATH®

En primer lugar se ha definido un modelo únicamente con las partes que resultan de interés para la simulación de un pathway eliminando los conceptos innecesarios de la compleja base de datos TRANSPATH®(véase anexo C). La figura 5.3 muestra —mediante una metáfora visual similar al diagrama de clases de UML— el modelo Ecore desarrollado. En el modelo encontramos la clase *Network*, elemento raíz del modelo. Nos referimos a esta clase como elemento *raíz* ya que podemos considerar éste modelo como un árbol si

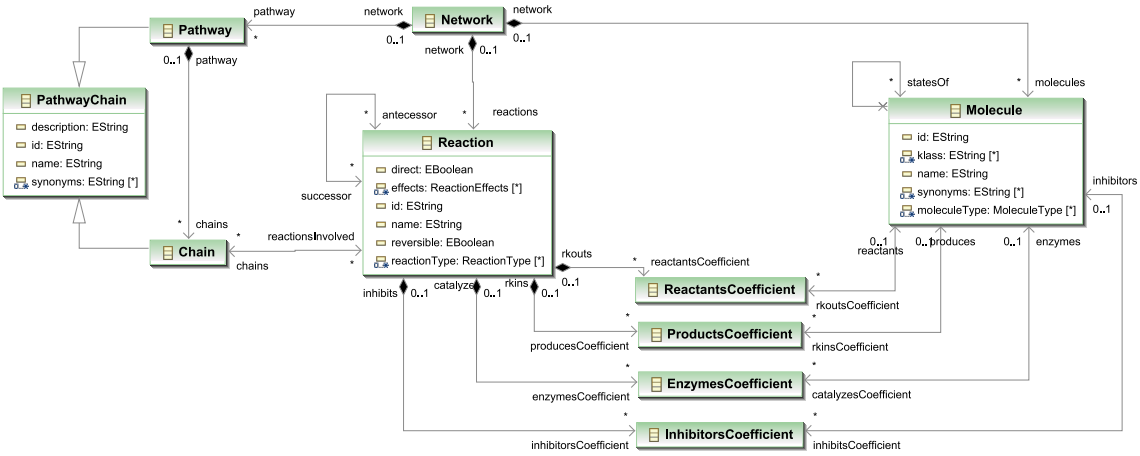


Figura 5.3: Modelo de la base de datos Transpath®.

navegamos hacia abajo las relaciones de contención partiendo del elemento network. Una Red —*Network*— contiene un conjunto de *Pathways*, *Reacciones* y *Moléculas*. A su vez, un *Pathway* se compone de diversas *Cadenas* —*Chains*— de reacciones, y una reacción puede estar involucrada en diferentes *cadenas*. Por último, las reacciones se relacionan con las moléculas. Una molécula puede ser un reactante de una reacción, puede ser un producto, o puede intervenir indirectamente como catalizador o inhibidor. Las clases *ReactantsCoefficient*, *ProductsCoefficient*, *EnzymesCoefficient* e *InhibitorsCoefficient* heredan todas de la clase *Coefficient*, omitida por motivos de claridad, que contiene un atributo *coefficient* de tipo entero. Estas clases permiten expresar que una molécula interviene con un cierto coeficiente[1] en una reacción, e intentan representar, según la expresividad de Ecore, la semántica de una clase asociación.

5.2.2. CPN Tools

La figura 5.4 muestra el modelo creado para la herramienta CPN Tools. En este caso, se ha decidido un diseño más alejado del diseño conceptual de una red de Petri Coloreada, incluyendo conceptos específicos de la propia herramienta *CPN Tools*. Esto se debe a que un diseño así que permite tratar con todos los conceptos interesantes de la herramienta *CPN Tools* desde EMF (color de los elementos gráficos y arcos, posiciones, etc.). Además, de esta manera el proceso de persistencia al fichero XML final es trivial, siendo una correspondencia 1-a-1.

El modelo —donde la clase raíz es *Cpnet*— se compone de dos grande bloques de clases: aquellas que cuelgan de *Globbox* y aquellas que lo hacen de *Page*. La figura 5.4 muestra estos dos grande grupos separados mediante una línea discontínua. El primer grupo

[1]El coeficiente representa el número de moléculas que aparecen en la ecuación de una reacción. Por ejemplo, en la reacción $2H_2 + O_2 = 2\ H_2O$, los coeficientes son los números que aparecen a la izquierda de las móleculas H_2 y H_2O.

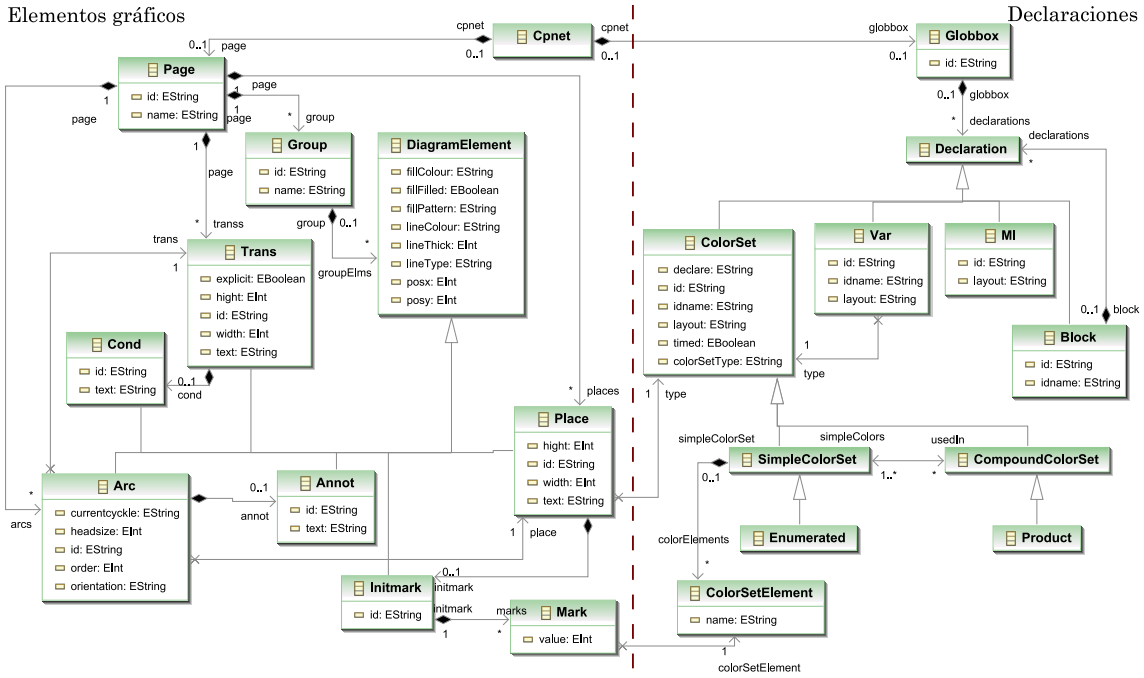


Figura 5.4: Modelo de la herramienta CPNTools.

de clases permite representar las declaraciones de la red (colores —*Color sets*—, *variables*, *bloques*, etc.). La herramienta permite definir distintos tipos de colores (*Color Sets*). De todos los tipos que proporciona, por simplicidad (ya que son los únicos necesarios) sólo se han incluido el *Color Set* simple «*Enumerated*» y el *Color Set* compuesto «*Product*».

El segundo grupo de clases (aquellas contenidas en el elemento *Page*) representan a todos los elementos visuales de la red de Petri coloreada. Estos elementos visuales heredan todos de la clase *DiagramElement*, y además, pueden estar contenidos en distintos grupos —*Group*—. Así, dentro de una *página* podemos encontrar lugares —*Places*—, transiciones —*Trans*—, arcos —*Arcs*—, anotaciones —*Annot*—, etc. Por su parte, se dice que los *lugares* que se han definido son de un tipo (color) que debe haber sido definido en las declaraciones (mediante el rol *type* desde la clase *Place* a la clase *ColorSet*). Las clases *InitMark* y *Mark* permiten representar el estado de un determinado *lugar*, indicando los tokens que se encuentran en éste. El tipo de estos tokens viene dado por el rol *colorSetElement* entre las clases *Mark* y *ColorSetElement*.

5.3. Proceso de transformación.

5.3.1. Preproceso

La fase de preproceso de los datos extrae los datos de un *pathway* de la base de datos TRANSPATH®como una instancia en XML en el formato descrito por el anexo C y lo convierte en una instancia en XMI que conforma al metamodelo de TRANSPATH®descrito en el apartado 5.2.1. Para realizar esta conversión se ha realizado una adaptación del código escrito en [76], resultando el código listado en el anexo I.

El preprocesador hace uso de la librería de procesamiento de documentos XML JDOM [56] para recorrer la instancia en XML de los datos a transformar. El proceso de traducción se realiza en dos pasos: En primer lugar, recorre el documento XML obteniendo la información acerca de los pathways, cadenas, reacciones y moléculas que se encuentran en el documento, y para cada uno de estos elementos, crea un objeto en EMF que sea instancia de su correspondiente *EClass*. Los atributos que sean de interés para este objeto son consultados en el árbol XML y establecidos en el objeto EMF que es almacenado en una estructura de datos temporal. En segundo lugar, se recorren los elementos creados y se establecen las referencias que relacionan unos elementos con otros (*Chains* con *Reaction*, *Reactions* con *Molecules*, etc.). Al finalizar este segundo paso ya se tiene una instancia del espacio tecnológico EMF en memoria que puede ser persistida a disco. El anexo E muestra el resultado de este proceso.

5.3.2. Transformación en QVT–Relations

Finalmente, se han definido las reglas de transformación que permiten convertir los datos del dominio origen al dominio destino. Estas reglas, expresan las correspondencias establecidas por los biólogos —cuando construyen una red de Petri coloreada de forma manual— entre los datos de TRANSPATH®y las primitivas de la herramienta *CPN Tools*. La tabla 5.1 muestra de forma simplificada estas correspondencias entre el dominio origen y el dominio destino.

El lenguaje con el que se expresan las relaciones entre ambos dominios es QVT-Relations, que se explicó en detalle en la sección 3.1.4 y que se resume nuevamente de forma breve a continuación. En QVT-Relations, una transformación son un conjunto de relaciones establecidas entre los dominios participantes en la transformación que deben cumplirse para que ésta sea satisfactoria [49]. Un dominio es una variable tipada que puede corresponderse con algún elemento del modelo que va a transformarse. Éste puede tener un *patrón*, que puede considerarse como un conjunto de restricciones que deben cumplir los elementos del modelo candidato —modelo sobre el que se aplica la transformación— para que se trate de una correspondencia válida.

Los dominios además, pueden caracterizarse mediante el uso de las palabras clave `checkonly` y `enforce`. Para un dominio `checkonly`, la transformación comprobará que

Transpath	**CPN Tools**
Network	Cpnet
Pathway	Globbox Page
Molecule (complex)	Product
Molecule (simple)	Enumerated
Reaction	Trans
Molecule (reactant)	Place Arc (de Place a Trans)
Molecule (product)	Place Arc (de Trans a Place)

Tabla 5.1: Correspondencias entre el dominio origen y el dominio destino.

existe una correspondencia válida —que satisfaga el patrón del dominio— en el modelo candidato. En caso de que el dominio destino sea `enforce`, si al ejecutar la transformación no existe ninguna correspondencia posible, se creará un elemento que cumpla con el patrón del dominio.

La *relación ReactantsToPlaces* 5.1, muestra un ejemplo de la sintaxis para la definición de *relaciones* y *dominios*. Esta *relación* establece la correspondencia entre una *molécula* y un *lugar*. Encontramos dos dominios: *tpDomain* se corresponde con el dominio origen (TRANSPATH®) y *cpnDomain* con el dominio destino (*CPN Tools*). Para el dominio *tpDomain* se comprueba que exista una molécula con un atributo, `name`, de tipo *String*. Para cada molécula que cumpla este patrón deberá existir (y si no existe, se creará) un *Place* en el dominio destino (*cpnDomain*), cuyo atributo `id` contenga el nombre de dicha molécula.

```
1       relation ReactantsToPlaces {
2         molecName1 : String;
3         checkonly domain tpDomain molecule1 : Molecule
4           {  name = molecName1  };
5
6         enforce domain cpnDomain place1 : Place
7           {  id = molecName1  };
8         //...
9         when
10          {  IsSimpleMolecule(molecule1);  }
11        where
12          {  ReactantsToArcs(molecule1,place1,...);  }
13      }
```

Listado 5.1: Regla *ReactantsToPlaces*.

A la aplicación de una regla pueden añadírsele pre- y post-condiciones mediante el uso de las cláusulas `when` y `where`. Por ejemplo, la regla *ReactantsToPlaces* sólo se aplicará en caso de que la molécula sea de tipo simple —IsSimpleMolecule(...) es una función que mediante una expresión OCL comprueba si la molécula es simple o compuesta—. La

cláusula `where` establece que tras la aplicación de la *relation* se procederá a la ejecución de la regla *ReactantsToArcs*.

El proceso de transformación completo se realiza de arriba a abajo, navegando el modelo origen a través de las relaciones de contención, definidas como asociaciones de composición. Esto es, se parte del elemento raíz del modelo origen (*Network*), y se navega hacia abajo (*Network* → *Pathways* → *Chains* → *Reactions* → *Coefficients* → *Molecules*) creando en el modelo destino los elementos correspondientes, tal y como expresa de forma resumida la tabla 5.1.

Para las declaraciones, la transformación creará un *ColorSet* de tipo *Enumerated* a partir de cada molécula simple. A partir de las moléculas complejas se crearán los *ColorSet* de tipo *Product*, y que estarán formados por los *Enumerated ColorSets* correspondientes a las moléculas simples que los forman.

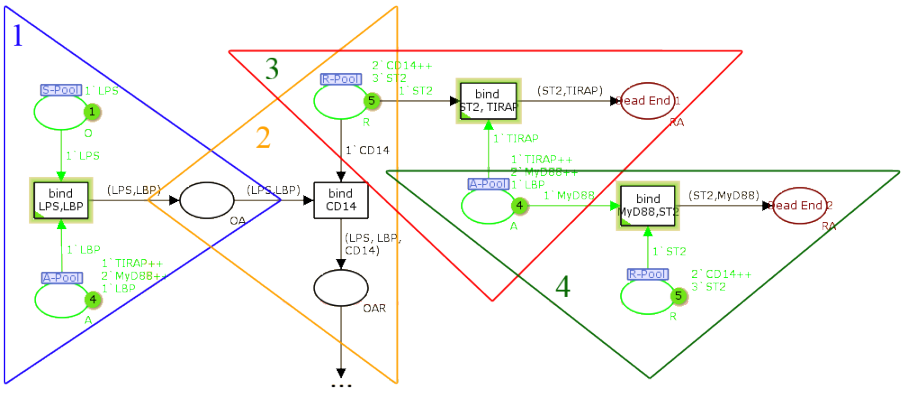


Figura 5.5: Representación parcial del Pathway del TLR4 en *CPN Tools*.

Para los elementos visuales se parte de un objeto de tipo *Reaction*. Para cada reacción, se crea un objeto de tipo *Trans*. Navegando el rol *reactantsCoefficient* de la clase *Reaction* se obtienen los reactantes. Para cada uno de ellos, se crea un objeto *Place*. Por último ya es posible enlazar cada *place* recién creado con el elemento *Trans* correspondiente mediante un arco —*Arc*— de tipo *PtoT* (*Place to trans*, según la terminología de *CPN Tools*). Para los productos de la reacción se procede de forma similar, en este caso navegando el rol *productsCoefficient*. Como resultado del proceso de transformación podemos observar la figura 5.5, que muestra la representación de las reacciones del caso de estudio en *CPN Tools*. En la figura se pueden observar cuatro triángulos numerados. Cada triángulo encierra los elementos generados para la reacción identificada con el mismo número. De esta manera, para la reacción (1) —LPS + LBP ⇋ LPS:LBP—, se han generado los elementos dentro del triángulo izquierdo (1). De igual manera ocurre con los triángulos 2, 3 y 4, que se corresponden con las consiguientes reacciones.

En la sección 6.3 se explican en detalle todas las regas especificadas.

5.3.3. Postproceso

Una vez se ha obtenido la instancia en XMI de la red de Petri Coloreada que permite representar el *pathway* en la herramienta de *CPN Tools* se puede proceder al paso 3 de la migración de datos. Éste toma como entrada una instancia en XMI del metamodelo Ecore mostrado en 5.4 y lo transforma en un documento XML utilizable por *CPN Tools*. En este paso además, se deben realizar operaciones adicionales, como calcular las posiciones que los elementos de la red de Petri tendrán sobre el *canvas* para que éstos se dibujen correctamente y no se solapen.

Capítulo 6

Implementación.

En el capítulo anterior, «Un enfoque de DSDM en la migración de datos biológicos», se ha descrito la aproximación propuesta, así como sus principales participantes y arquitectura genérica. A lo largo de las siguientes páginas se describirá como se ha implementado esta propuesta hasta la obtención de un prototipo final ejecutable.

Las siguientes secciones describirán, en primer lugar, cómo se han implementado definido e implementado los metamodelos que participan en el proceso de transformación. En segundo lugar, se presentarán las herramientas (plugins) que permiten a EMF interoperar de forma transparente con los documentos en los formatos nativos que emplean las plataformas origen y destino (TRANSPATH®y CPN Tools). En tercer lugar se presentarán las reglas de transformación definidas para la ejecución de las transformaciones en las dos plataformas de transformaciones de modelos seleccionadas. Por último se presentarán un conjunto de plugins adicionales que se han implementado para extender la funcionalidad básica de la herramienta MediniQVT, haciendo la invocación de transformaciones más sencilla para el usuario final.

6.1. Metamodelos

En esta sección se describen los plugins que se han creado a partir de los metamodelos para poder tratar con instancias de estos modelos Ecore directamente en Eclipse, sin necesidad de registrar de forma manual los modelos en EMF. La mayor parte de código de estos plugins se ha generado de forma automática, así que aquellas clases que se hayan obtenido mediante técnicas de programacióin generativas serán simplemente listadas y descritas a alto nivel de abstracción. Las partes de código que se hayan modificado o escrito a mano se describirán en mayor detalle.

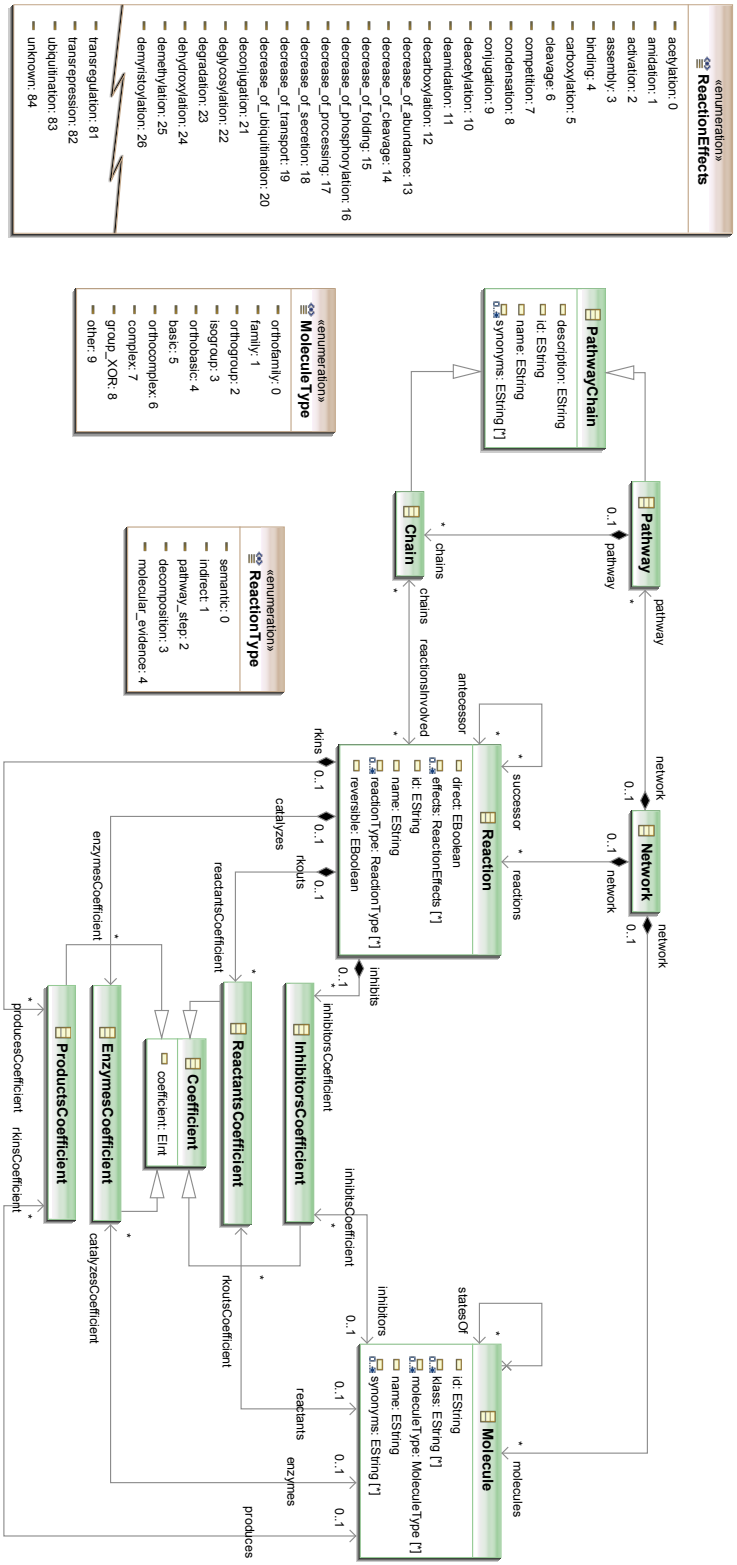


Figura 6.1: Modelo Ecore completo de la base de datos TRANPATH®.

6.1.1. Plugin es.upv.dsic.issi.moment.intergenomics.transpath

6.1.1.1. Descripción

El plugin es.upv.dsic.issi.moment.intergenomics.transpath define el metamodelo para la base de datos TRANSPATH®que será empleado en las transformaciones. Fundamentalmente, está formado por código generado automáticamente por EMF. Para ello, se ha definido mediante Ecore el metamodelo que se muestra en la figura 6.1 y se han empleado los mecanismos estándares de EMF.

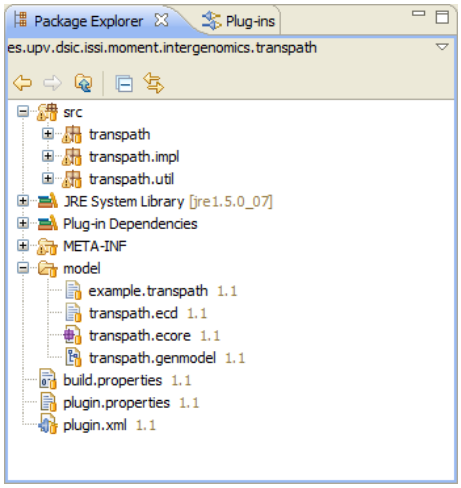


Figura 6.2: Proyecto es.upv.dsic.issi.moment.intergenomics.transpath.

6.1.1.2. Dependencias

El código de este plugion está generado completamente por EMF, sin haberse modificado en absoluto, por ello las las dependencias de otros plugins de Eclipse son únicamente las habituales:

- org.eclipse.core.runtime
- org.eclipse.emf.ecore

6.1.1.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.moment.intergenomics.transpath

El paquete es.upv.dsic.issi.moment.intergenomics.transpath contiene las interfaces que implementan las clases que se encuentran en el paquete es.upv.dsic.issi.moment.intergenomics.transpath.impl. En EMF se implementa tanto una interfaz como una clase por cada una de las clases definidas en el modelo Ecore. De esta manera, se pueden simular en Java los mecanismos de herencia múltiple que permite Ecore. Las interfaces TranspathFactory y TranspathPackage no se generan a partir de las clases del modelo, sino que se corresponden con el modelo en sí. En particular, TranspathFactory es la interfaz que permite acceder a la factoría para crear nuevas instancias de los objetos del modelo, y TranspathPackage se corresponde con la interfaz que contiene la información del paquete (nsPrefix, nsUri, etc.) así como el acceso a la instancia singleton del paquete.

Listado de clases

- Chain.java
- Coefficient.java
- EnzymesCoefficient.java
- InhibitorsCoefficient.java
- Molecule.java
- MoleculeType.java
- Network.java
- Pathway.java
- PathwayChain.java
- ProductsCoefficient.java
- ReactantsCoefficient.java
- Reaction.java
- ReactionEffects.java
- ReactionType.java
- TranspathFactory.java
- TranspathPackage.java

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.impl

El paquete es.upv.dsic.issi.moment.intergenomics.transpath.impl contiene las clases que implementan el código del modelo. En este sentido, se crea una clase Java (con el sufijo *–Impl*) para cada una de las clases del modelo. Cada una de esta clases contiene los métodos (con código generado) para consultar, navegar y modificar instancias del modelo (métodos *getters* y *setters*). Estos métodos además, contienen los mecanismos pertinentes para mantener la coherencia entre los objetos del modelo cuando se encuentran relaciones bidireccionales entre las clases del modelo (mediante la interfaz *Notifier*).

Además de aquellas clases que se corresponden con las del modelo, encontramos las clases especiales TranspathFactoryImpl y TranspathPackageImpl. La primera de ellas se corresponde con la clase que implementa las factorías de objetos del modelo (en EMF se recomienda usar el patrón de factorías, en lugar de emplear el operador *new* de Java). La segunda clase, TranspathPackageImpl, se corresponde con la clase que implementa el paquete del modelo. De ésta clase sólo habrá una única instancia en memoria (según el patrón *singleton*) y contiene los métodos que permiten recrear en memoria la estructura del modelo al inicializarse el plugin.

Listado de clases

- ChainImpl.java
- CoefficientImpl.java
- EnzymesCoefficientImpl.java
- InhibitorsCoefficientImpl.java
- MoleculeImpl.java
- NetworkImpl.java
- PathwayChainImpl.java
- PathwayImpl.java
- ProductsCoefficientImpl.java
- ReactantsCoefficientImpl.java
- ReactionImpl.java
- TranspathFactoryImpl.java
- TranspathPackageImpl.java

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.util

En este paquete se encuentran únicamente dos clases de utilidades que realizan tareas auxiliares comunes en los plugins de EMF para la creación y navegación de los elementos del modelo.

Listado de clases

- TranspathAdapterFactory.java
- TranspathSwitch.java

6.1.2. Plugin es.upv.dsic.issi.moment.intergenomics.transpath.edit

6.1.2.1. Descripción

El plugin es.upv.dsic.issi.moment.intergenomics.transpath.edit proporciona la funcionalidad básica para representar los elementos del modelo de *TRANSPATH*® en un editor.

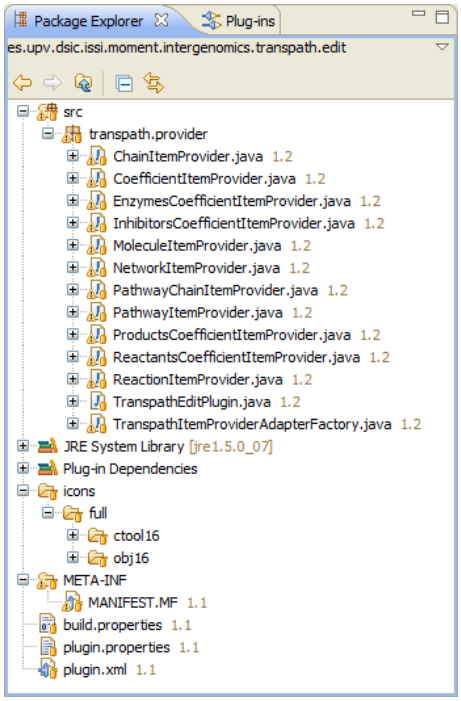


Figura 6.3: Proyecto es.upv.dsic.issi.moment.intergenomics.transpath.edit.

6.1.2.2. Dependencias

Las dependencias de este plugin son:

- org.eclipse.core.runtime (por ser un plugin de Eclipse)
- org.eclipse.emf.edit (por ser un plugin de EMF Edit)
- El metamodelo de TRANSPATH®(es.upv.dsic.issi.moment.intergenomics.transpath)
- por ser el soporte para su edición.

6.1.2.3. Descripción de paquetes y clases

El contenido de este plugin se ha generado de forma automática, aunque se han realizado cambios menores en el código (principalmente en los métodos getText() de los diferentes *ItemProviders*).

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.provider

Este es el único paquete del plugin. Contiene la clase activadora del plugin (TranspathEditPlugin), que se encarga de controlar el ciclo de vida del plugin mediante su instancia *singleton*. A su vez, contiene la clase TranspathItemProviderAdapterFactory que es la clase de la factoría para acceder a cada uno de los *items providers* contenidos en el paquete.

Por último, contiene una clase *provider* por cada clase del modelo (todas con el sufijo *–ItemProvider*). Estas clases sirven para consultar cómo se deben mostrar los elementos en los editores, esto es, qué icono representará a cada elemento, cómo se construirá el texto de sus etiquetas, qué elementos del modelo son sus *hijos* (de manera que se presentan los correspondientes menús de creación de *nuevo hijo*), qué propiedades deben de mostrarse en las hojas de propiedades para poder ser visualizadas/editadas, etc.

Listado de clases y métodos

A continuación se listan las clases del paquete. Además, se listan los métodos de la clase ChainItemProvider como ejemplo de los métodos proporcionados por un *ItemProvider*.

- ChainItemProvider.java
  - ChainItemProvider(AdapterFactory)
  - getPropertyDescriptors(Object)
  - addReactionsInvolvedPropertyDescriptor(Object)
  - getImage(Object)
  - getText(Object)
  - notifyChanged(Notification)
  - collectNewChildDescriptors(Collection, Object)
  - getResourceLocator()
- CoefficientItemProvider.java
- EnzymesCoefficientItemProvider.java
- InhibitorsCoefficientItemProvider.java
- MoleculeItemProvider.java
- NetworkItemProvider.java
- PathwayChainItemProvider.java
- PathwayItemProvider.java
- ProductsCoefficientItemProvider.java
- ReactantsCoefficientItemProvider.java
- ReactionItemProvider.java
- TranspathEditPlugin.java
- TranspathItemProviderAdapterFactory.java

6.1.3. Plugin es.upv.dsic.issi.moment.intergenomics.transpath.editor

6.1.3.1. Descripción

Este plugin implementa un editor en árbol típico de EMF. Se encuentra asociado por defecto con la extensión «*.transpath». Además, incluye el asistente de creación de un

nuevo fichero «*.transpath» integrado con el resto de asistentes de creación de recursos de Eclipse.

6.1.3.2. Dependencias

El plugin es.upv.dsic.issi.moment.intergenomics.transpath.editor tiene las siguientes dependencias:

- org.eclipse.core.runtime
- org.eclipse.core.resources
- org.eclipse.ui.ide
- org.eclipse.emf.ecore.xmi
- org.eclipse.emf.edit.ui
- es.upv.dsic.issi.moment.intergenomics.transpath.edit

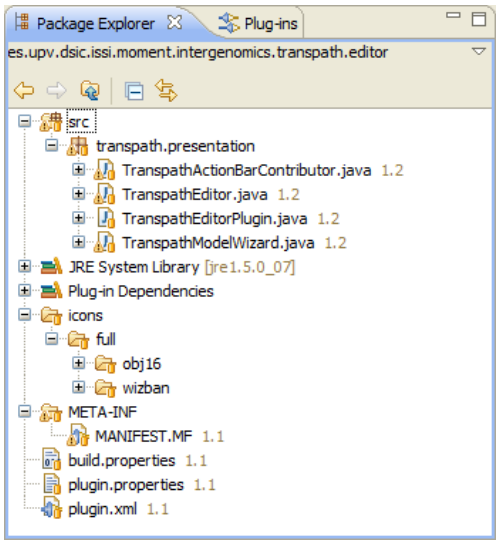


Figura 6.4: Proyecto es.upv.dsic.issi.moment.intergenomics.transpath.editor.

6.1.3.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.presentation

Este es el único paquete del plugin y su código es completamente generado de forma automática. Contiene a su vez las siguiente clases:

TranspathActionBarContributor

La clase TranspathActionBarContributor contiene las acciones que se contribuirán a la interfaz estándar de Eclipse. Como contribuciones entendemos los botones que se añadirán a la barra de herramientas de Eclipse, el menú que aparecerá en la barra de menús cuando el editor se encuentra abierto y activo, etc.

TranspathEditor

La clase TranspathEditor implementa el editor en árbol para las instancias del modelo de *TRANSPATH®*. En cuanto a funcionalidad es similar a cualquier otro editor en árbol genérico, siendo su única diferencia en cómo en la inicialización del editor (método initializeEditingDomain()) se añade el *ItemProviderAdapterFactory* (creado en el plugin es.upv.dsic.issi.moment.intergenomics.transpath.edit comentado anteriormente) al adapterFactory propio del editor.

```
1       //...
2       adapterFactory = new ComposedAdapterFactory(ComposedAdapterFactory.Descriptor.
            Registry.INSTANCE);
3
4       adapterFactory.addAdapterFactory(new ResourceItemProviderAdapterFactory());
5       adapterFactory.addAdapterFactory(new TranspathItemProviderAdapterFactory());
6       adapterFactory.addAdapterFactory(new ReflectiveItemProviderAdapterFactory());
7       //...
```

Listado 6.1: Fragmento de código del método initializeEditingDomain() de la clase TranspathEditor.

TranspathEditorPlugin

La clase TranspathEditorPlugin es la clase activadora del plugin conteniendo la instancia *singleton* que permite acceder al estado de éste y controlar su ciclo de vida.

TranspathModelWizard

Esta clase implementa el asistente de creación de una nueva instancia del modelo de *TRANSPATH*®. El asistente se integra en la categoría «Example EMF Model Creation Wizards».

6.1.4. Plugin es.upv.dsic.issi.moment.intergenomics.cpn

6.1.4.1. Descripción

El plugin es.upv.dsic.issi.moment.intergenomics.cpn define el metamodelo para la herramienta CPN Tools que será empleado en las transformaciones. Fundamentalmente, está formado por código generado automáticamente por EMF.

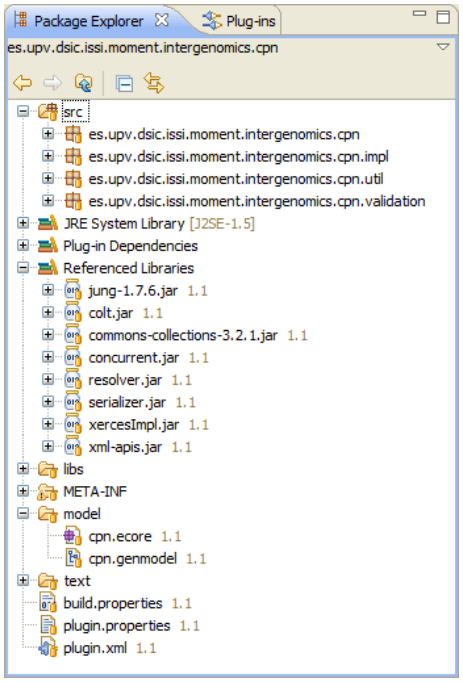


Figura 6.5: Proyecto es.upv.dsic.issi.moment.intergenomics.cpn.

Para ello, se ha definido mediante Ecore el metamodelo de CPN Tools que se muestra en la figura 6.6. Se puede observar en la clase Page que se ha declarado el método void performLayout(). En este caso, EMF genera la declaración del método, pero no su implementación. De esta manera, el comportamiento de este método es el único código escrito de forma manual en todo el proyecto. En concreto, este método permite organizar de forma automática los elementos gráficos que existen en una página mediante un algoritmo de *layout*. En el punto 6.1.4.3 se describe en mayor detalle la función que realiza este método.

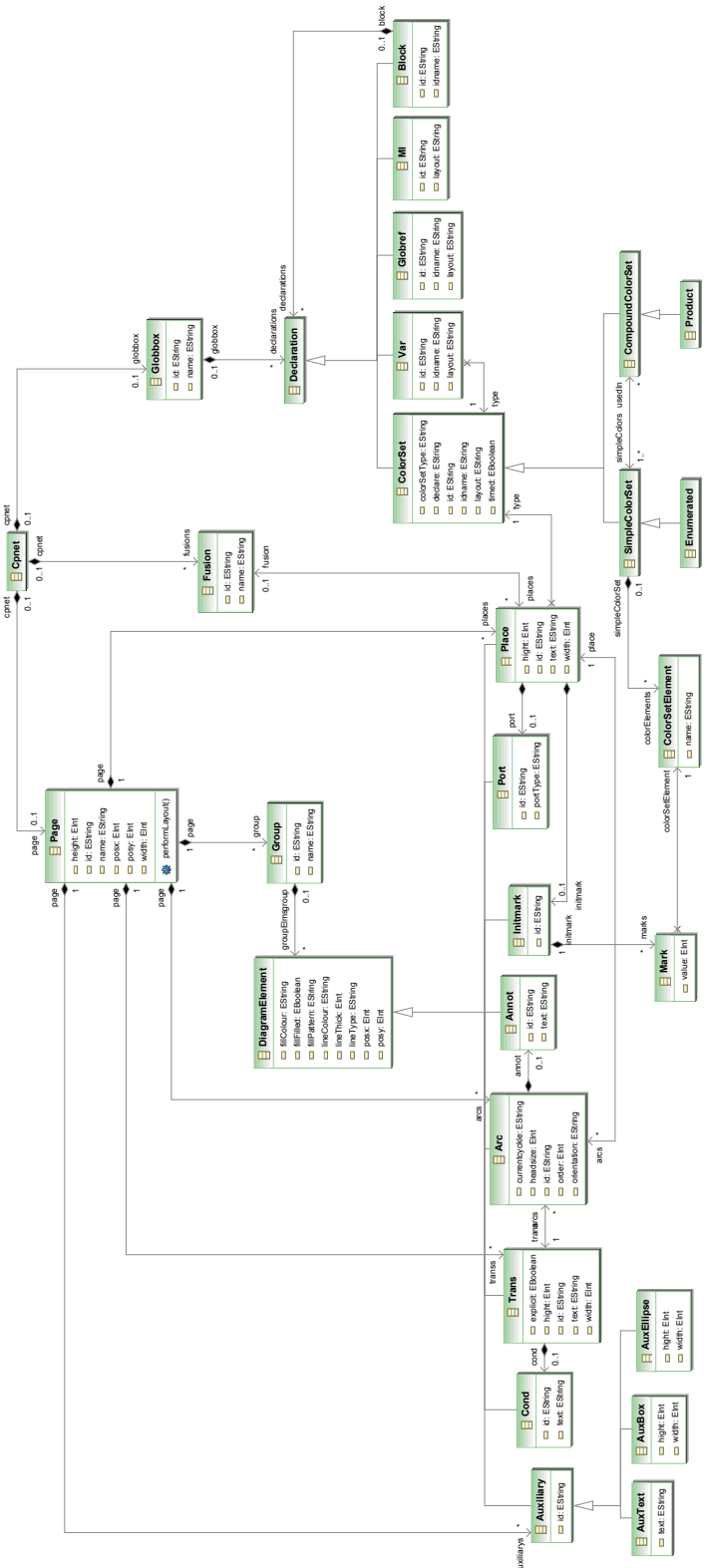


Figura 6.6: Modelo Ecore completo de la herramienta *CPN Tools*.

6.1.4.2. Dependencias

Las dependencias de este plugin se pueden clasificar en dependencias de otros plugins de Eclipse, y dependencias de otras librerías genéricas de Java. Respecto a las dependencias de otros plugins de Eclipse, puesto que se trata de un plugin de código generado de EMF depende de los siguientes plugins, como es habitual:

- org.eclipse.core.runtime
- org.eclipse.emf.ecore

Respecto a otras librerías de Java, puesto que se ha implementado código de el método performLayout() de la clase Page de forma manual, se ha necesitado incluir las bibliotecas de las que se hace uso para realizar el redibujado de los elementos gráficos. En este sentido, se ha hecho uso de la librería JUNG 1.7.6 (Java Universal Network/Graph Framework [69]), una librería de código abierto que implementa diferentes algoritmos de redibujado.

Esta biblioteca (archivo `jung-1.7.6.jar`) tiene como dependencias:

- **Colt [20]**. Una biblioteca de alto rendimiento para computación científica y técnica implementada en Jqava. Este paquete a su vez, requiere la biblioteca util.concurrent [41].

  Paquetes requeridos:

  - `colt.jar`
  - `concurrent.jar`

- **Apache Commons Collections [22]**. Una API de Apache que extiende el uso de colecciones en Java.

  Paquetes requeridos:

  - `commons-collections-3.2.1.jar`

- **Apache Xerces 2 [24]**. Un conjunto de librerías Java de Apache para la construcción de parser para documentos XML.

  Paquetes requeridos:

  - `resolver.jar`
  - `serializer.jar`
  - `xercesImpl.jar`
  - `xml-apis.jar`

6.1.4.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.moment.intergenomics.cpn

El paquete es.upv.dsic.issi.moment.intergenomics.cpn contiene las interfaces que implementan las clases que se encuentran en el paquete es.upv.dsic.issi.moment.intergenomics.cpn.impl. En EMF se implementa tanto una interfaz como una clase por cada una de las clases definidas en el modelo Ecore. De esta manera, se pueden simular en Java los mecanismos de herencia múltiple que permite Ecore. Las interfaces CpnFactory y CpnPackage no se generan a partir de las clases del modelo, sino que se corresponden con el modelo en sí. En particular, CpnFactory es la interfaz que permite acceder a la factoría para crear nuevas instancias de los objetos del modelo, y CpnPackage se corresponde con la interfaz que contiene la información del paquete (nsPrefix, nsUri, etc.) así como el acceso a la instancia singleton del paquete.

Listado de clases

- Annot.java
- Arc.java
- AuxBox.java
- AuxEllipse.java
- Auxiliary.java
- AuxText.java
- Block.java
- ColorSet.java
- ColorSetElement.java
- CompoundColorSet.java
- Cond.java
- Cpnet.java
- CpnFactory.java
- CpnPackage.java
- Declaration.java
- DiagramElement.java
- Enumerated.java
- Fusion.java
- Globbox.java
- Globref.java
- Group.java
- Initmark.java
- Mark.java
- Ml.java
- Page.java
- Place.java
- Port.java
- Product.java
- SimpleColorSet.java
- Trans.java
- Var.java

Paquete es.upv.dsic.issi.moment.intergenomics.cpn.impl

El paquete es.upv.dsic.issi.moment.intergenomics.cpn.impl contiene las clases que implementan el código del modelo. En este sentido, se crea una clase Java (con el sufijo *–Impl*) para cada una de las clases del modelo. Cada una de esta clases contiene los métodos (con código generado) para consultar, navegar y modificar instancias del modelo (métodos *getters* y *setters*). Estos métodos además, contienen los mecanismos pertinentes para mantener la coherencia entre los objetos del modelo cuando se encuentran relaciones bidireccionales entre las clases del modelo (mediante la interfaz *Notifier*).

Además de aquellas clases que se corresponden con las del modelo, encontramos las clases especiales CpnFactoryImpl y CpnPackageImpl. La primera de ellas se corresponde con la clase que implementa las factorías de objetos del modelo (en EMF se recomienda usar el patrón de factorías, en lugar de emplear el operador *new* de Java). La segunda clase, CpnPackageImpl, se corresponde con la clase que implementa el paquete del modelo. De ésta clase sólo habrá una única instancia en memoria (según el patrón *singleton*) y contiene los métodos que permiten recrear en memoria la estructura del modelo al inicializarse el plugin.

Listado de clases

- AnnotImpl.java
- ArcImpl.java
- AuxBoxImpl.java
- AuxEllipseImpl.java
- AuxiliaryImpl.java
- AuxTextImpl.java
- BlockImpl.java
- ColorSetElementImpl.java
- ColorSetImpl.java
- CompoundColorSetImpl.java
- CondImpl.java
- CpnetImpl.java
- CpnFactoryImpl.java
- CpnPackageImpl.java
- DeclarationImpl.java
- DiagramElementImpl.java
- EnumeratedImpl.java
- FusionImpl.java
- GlobboxImpl.java
- GlobrefImpl.java
- GroupImpl.java
- InitmarkImpl.java
- MarkImpl.java
- MlImpl.java
- PageImpl.java
- PlaceImpl.java
- PortImpl.java
- ProductImpl.java
- SimpleColorSetImpl.java
- TransImpl.java
- VarImpl.java

Implementación del método es.upv.dsic.issi.moment.intergenomics.cpn.impl.-PageImpl.performLayout().

El método performLayout() tiene la siguiente signatura:

```
1       public boolean performLayout(Integer width, Integer height, Integer steps);
```

Listado 6.2: Signatura del método performLayout(...)

Donde *width* y *height* se corresponde con el tamaño total del *canvas* sobre el que se dibujará el grafo. De esta manera, al aplicar el algorimo de redibujado se limitará la posición de los elementos del grafo al espacio encerrado por estas dimensiones. El argumento *steps* define el número de iteraciones del algoritmo de *layout*. Dependiendo del número de iteraciones se obtendrán mejores o peores resultados, ya que el algoritmo no es detemrinista.

Para la implementación del algortimo de redibujado del grafo se ha utilizado la biblioteca JUNG [69], que entre otra funcionalidad, proporciona diversos algoritmos de reposicionamiento de los nodos de un grafo. En este caso, vamos a emplear el algoritmo de *Fruchterman-Reingold* para el redibujado de grafos dirigidos.

Este algoritmo de *layout* está implementado en la clase edu.uci.ics.jung.visualization.-FRLayout, que recibe como argumento del constructor el grafo a redibujar. Este grafo debe representarse en términos de las clases proporcionadas por la librería JUNG, por tanto, el primer paso que debemos realizar en este método es encapsular los elementos gráficos de nuestro modelo en los elementos gráficos propios de JUNG. La figura 6.7 muestra las interfaces de la API de JUNG que permite representar diferentes tipos de grafos. En la figura 6.8 encontramos las clases que emplearemos en nuestro código, así como se relacionan con las interfaces de la figura 6.7.

La clase que representa un grafo dirigido en JUNG es edu.uci.ics.jung.graph.impl.DirectedSparseGraph, a la que se le añaden los distintos nodos y arcos mediante los correspondientes métodos addVertex(Vertex vertex) y addEdge(Edge edge). En este caso, puesto que vamos a construir un grafo dirigido, nuestras instancias de Edge y Vertex serán de tipo DirectedSparseEdge y DirectedSparseEdge.

No obstante, como queremos que al aplicar el algoritmo de redibujado se reposicionen los elementos de la red de Petri, extenderemos estas clases para permitir al algoritmo de *layout* tratar con los elementos propios del diagrama (clase es.upv.dsic.issi.moment.intergenomics.cpn.DiagramElement) de nuestro modelo de *CPN Tools*. De esta manera, tras la aplicación del redibujado, se puede identificar unívocamente qué elementos del grafo se corresponden con qué elementos del diagrama. De esta manera, los nodos de nuestro grafo dirigido serán instancias de la clase personalizada CustomDirectedSparseVertex, tal y como se muestra en el listado 6.3.

```
1       class CustomDirectedSparseVertex extends DirectedSparseVertex {
2
3         private DiagramElement element;
4
```

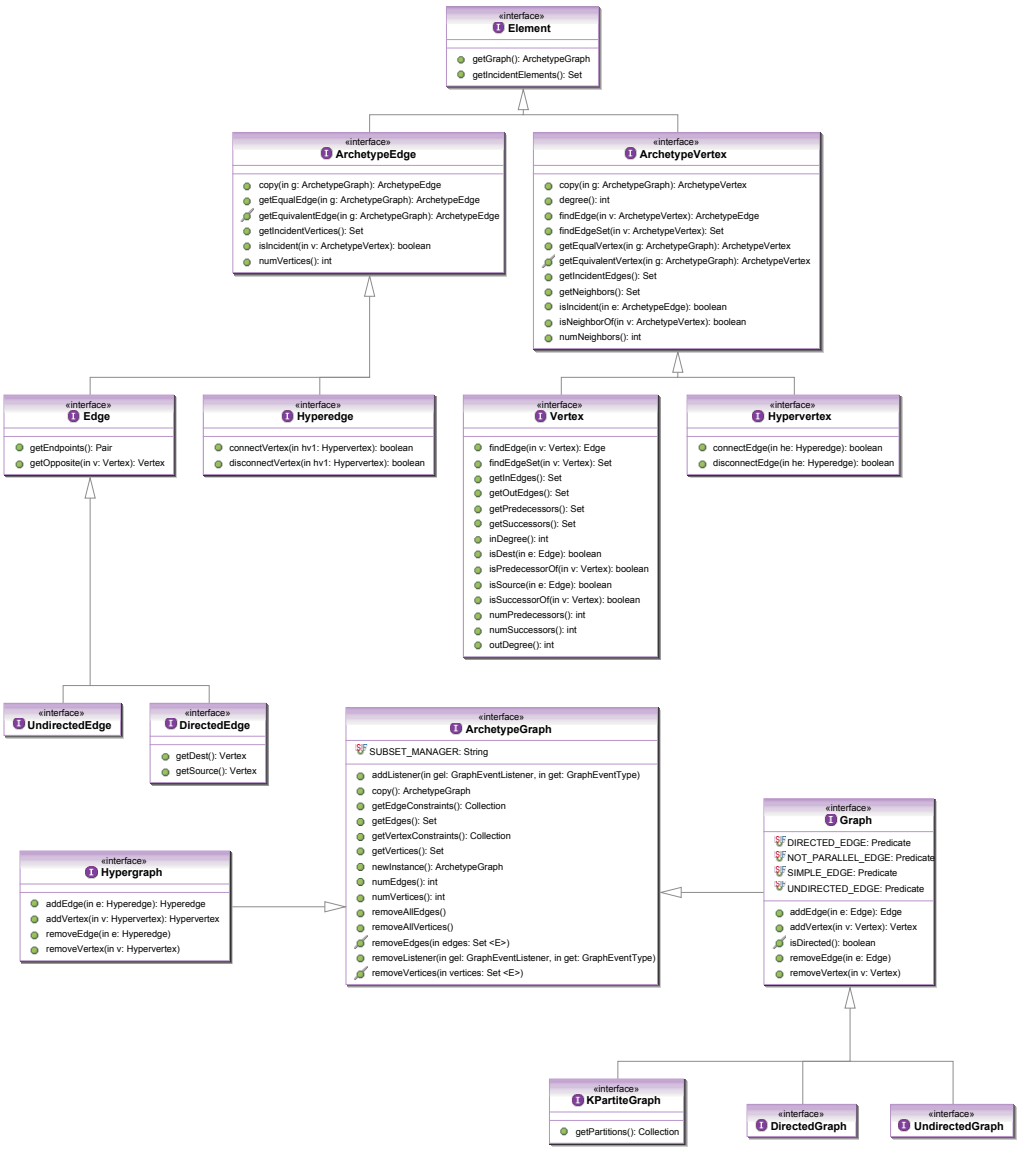


Figura 6.7: Subconjunto de interfaces de la librería JUNG.

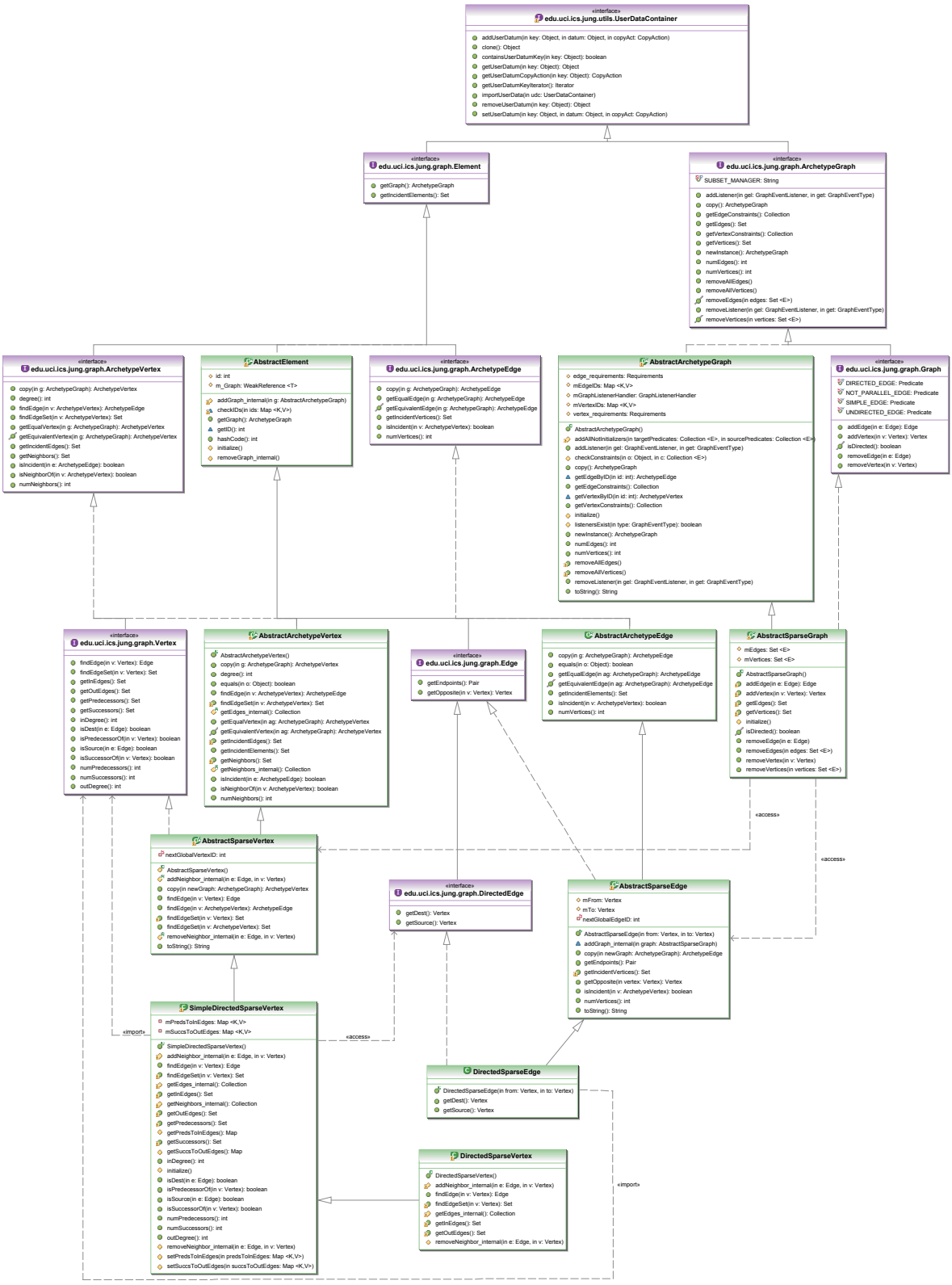


Figura 6.8: Subconjunto de clases de la librería JUNG.

```
5         public CustomDirectedSparseVertex(DiagramElement element) {
6           super();
7           this.element = element;
8         }
9
10        public void setLocation(int x, int y) {
11          getElement().setPosx(x);
12          getElement().setPosy(y);
13        }
14
15        public DiagramElement getElement() {
16          return element;
17        }
18      }
```

Listado 6.3: Clase es.upv.dsic.issi.moment.intergenomics.cpn.impl.PageImpl.performLayout(...).-CustomDirectedSparseVertex

Se observa que la clase CustomDirectedSparseVertex encapsula el elemento del diagrama de quien heradan todos los elementos gráficos de la red de Petri, según el diagrama de clases de la figura 6.6. Este elemento gráfico se establece en el momento de creación del objeto al ser un argumento del constructor, y el método oid setLocation(int x, int y) modifica directamente la posición del elemento gráfico que representa el nodo del grafo.

Para los arcos de nuestro grafo se ha extendido la clase DirectedSparseEdge mediante la case CustomDirectedSparseEdge, tal y como se muestra en el listado 6.4.

```
1       class CustomDirectedSparseEdge extends DirectedSparseEdge {
2
3         private DiagramElement element;
4
5         public CustomDirectedSparseEdge(Vertex from, Vertex to, DiagramElement element) {
6           super(from, to);
7           this.element = element;
8         }
9
10        public void setLocation(int x, int y) {
11          getElement().setPosx(x);
12          getElement().setPosy(y);
13        }
14
15        public DiagramElement getElement() {
16          return element;
17        }
18      }
```

Listado 6.4: Clase es.upv.dsic.issi.moment.intergenomics.cpn.impl.PageImpl.performLayout(...).-CustomDirectedSparseEdge

En este caso se observa que el constructor, además de recibir el elemento del diagrama (que será una instancia de la clase *Arc*), recibe los vértices que relaciona dicho arco (que serán instancias de CustomDirectedSparseVertex).

Un vez se han definido las clases que encapsularán nuestros elementos gráficos de la red de Petri, se puede escribir el código que reconstruirá el grafo en términos de las clases de la librería JUNG, tal y como demuestra el listado de código 6.5.

```
1   DirectedSparseGraph graph = new DirectedSparseGraph();
2
3   for (Arc arc : this.getArcs()) {
4
5     DirectedSparseVertex vertexPlace = null;
6
7     for (Object obj : graph.getVertices()) {
8       CustomDirectedSparseVertex vertex =  (CustomDirectedSparseVertex) obj;
9       if (vertex.getElement().equals(arc.getPlace()))  {
10        vertexPlace = vertex;
11        break;
12      }
13    }
14    if (vertexPlace == null) {
15      vertexPlace = new CustomDirectedSparseVertex(arc.getPlace());
16      graph.addVertex(vertexPlace);
17    }
18
19    DirectedSparseVertex vertexTrans = null;
20
21    for (Object obj : graph.getVertices()) {
22      CustomDirectedSparseVertex vertex =  (CustomDirectedSparseVertex) obj;
23      if (vertex.getElement().equals(arc.getTrans()))  {
24        vertexTrans = vertex;
25        break;
26      }
27    }
28    if (vertexTrans == null) {
29      vertexTrans = new CustomDirectedSparseVertex(arc.getTrans());
30      graph.addVertex(vertexTrans);
31    }
32
33    CustomDirectedSparseEdge edge = null;
34
35    if (arc.getOrientation().equals("PtoT")) {
36      edge = new CustomDirectedSparseEdge(vertexPlace,vertexTrans, arc.getAnnot());
37      } else {
38        edge = new CustomDirectedSparseEdge(vertexTrans,vertexPlace, arc.getAnnot());
39      }
40      graph.addEdge(edge);
41    }
42
43    //...
```

Listado 6.5: Construcción del grafo a ser redibujado

En primer lugar, tal y como muestra el código (línea 1), se crea el objeto que representará a nuestro grafo que es instancia de la clase *DirectedSparseGraph*.

Una vez tenemos dicho objeto, al que añadiremos los diferentes arcos y nodos de la red de Petri, iteraremos sobre el conjunto de arcos de la red (línea 3). De esta manera, para cada arco de la red de Petri, que representa un enlace entre un elemento de tipo *Place* y un elemento de tipo *Trans*:

1. Dado el *Place* del arco, buscaremos si ya existe un *Vertex* que encapsule a dicho *Place* (7–13). Se debe comprobar esto puesto que un único *Place* puede estar relacionado con diferentes arcos.

   Si ya existe dicho *Vertex* (9), tomaremos nota de él en la variable vertexPlace (10). Sino (14), crearemos una nueva instancia (15) y la añadiremos al grafo (16).

2. Por su parte, con el *Trans* procederemos de la misma manera: buscaremos si ya existe un *Vertex* que encapsule a dicho *Trans* (21–27). Si ya existe dicho *Vertex* (23), tomaremos nota de él en la variable vertexTrans (24). Sino (28), crearemos una nueva instancia (29) y la añadiremos al grafo (30).

3. Por último, una vez hemos identificado/creado los nodos del grafo que relaciona el arco del diagrama, crearemos el arco del grafo (35–41).

   En el momento de creación del arco, el orden de los vértices (es dirigido) dependerá de si el arco del diagrama es de tipo `PtoT` (Place-To-Trans) o `TtoP` (Trans-To-Place). Finalmente, en la línea 40 se añade el arco al grafo.

Una vez se ha creado el grafo, se puede aplicar el algoritmo de redibujado tal y como muestra el listado 6.6.

```
1       FRLayout layout = new FRLayout(graph);
2       layout.initialize(new Dimension(width, height));
3
4       layout.setMaxIterations(steps);
5
6       while (!layout.incrementsAreDone()) {
7         layout.advancePositions();
8       }
```

Listado 6.6: Invocación del algoritmo de *Fruchterman-Reingold*

En primer lugar, se crea la instancia de *FRLayout*. Después se inicializa con la dimensión máxima que podrá adquirir como máximo el grafo. Por último, se establece el máximo de iteraciones que se pueden aplicar, y se invoca al método de reposicionamiento mientras no se haya alcanzado el máximo de pasos.

Cuando se termina de aplicar el algoritmo, la instancia de la clase *FRLayout* (variable layout) contiene las posiciones de cada uno de los elementos del grafo. De esta manera, recorreremos todos los elementos del grafo, y le preguntaremos al resultado del redibujado cuál es la posición que ocupa dicho elemento en el grafo (listado 6.7).

```
1       for (Object obj : graph.getEdges()) {
2         CustomDirectedSparseEdge edge = (CustomDirectedSparseEdge) obj;
3
4         CustomDirectedSparseVertex source = (CustomDirectedSparseVertex)edge.getSource();
5         CustomDirectedSparseVertex target = (CustomDirectedSparseVertex)edge.getDest();
6
7         int sourceX = (int)layout.getX(source)-(width/2);
8         int sourceY = (int)layout.getY(source)-(height/2);
9         int targetX = (int)layout.getX(target)-(width/2);
10        int targetY = (int)layout.getY(target)-(height/2);
11
12        source.setLocation(sourceX,sourceY);
13        target.setLocation(targetX,targetY);
14        edge.setLocation(sourceX+(targetX-sourceX)/2, sourceY+(targetY-sourceY)/2);
15
16      }
```

Listado 6.7: Post-proceso tras la aplicación de *Fruchterman-Reingold*

Como se observa en el código, se itera sobre todos los arcos del grafo, obteniendo los vértices origen y destino que relaciona dicho arco. En las variables sourceX, sourceY, targetX, targetY se almacena la posición de los nodos origen y destino. Las sustracciones «`-(width/2)`» y «`-(height/2)`» se hacen símplemente para centrar el origen de coordenadas de las posiciones.

Por último (12–14), se establece las posiciones tanto de los nodos como del arco. En este caso, la operación «`+(targetX-sourceX)/2`» (y «`+(targetY-sourceY)/2`») se realiza para que el extremo del arco apunte exactamente al centro de los *Trans* y los *Place* de la red de Petri final.

Paquete es.upv.dsic.issi.moment.intergenomics.cpn.util

En este paquete se encuentran únicamente dos clases de utilidades que realizan tareas auxiliares comunes en los plugins de EMF para la creación y navegación de los elementos del modelo.

Listado de clases

- CpnAdapterFactory.java
- CpnSwitch.java

Paquete es.upv.dsic.issi.moment.intergenomics.cpn.validation

Este paquete realmente no realiza ninguna acción y puede ignorarse. Simplemente declara una serie de interfaces que declaran interfaces de validación. Este código se crea como ejemplo de cómo se pueden extender las capacidades de generación de código de EMF.

Listado de clases

- AnnotValidator.java
- ArcValidator.java
- AuxBoxValidator.java
- AuxEllipseValidator.java
- AuxiliaryValidator.java
- AuxTextValidator.java
- BlockValidator.java
- ColorSetElementValidator.java
- ColorSetValidator.java
- CompoundColorSetValidator.java
- CondValidator.java
- CpnetValidator.java
- DeclarationValidator.java
- DiagramElementValidator.java
- EnumeratedValidator.java
- FusionValidator.java
- GlobboxValidator.java
- GlobrefValidator.java
- GroupValidator.java
- InitmarkValidator.java
- MarkValidator.java
- MlValidator.java
- PageValidator.java
- PlaceValidator.java
- PortValidator.java
- ProductValidator.java
- SimpleColorSetValidator.java
- TransValidator.java
- VarValidator.java

6.1.5. Plugin es.upv.dsic.issi.moment.intergenomics.cpn.edit

6.1.5.1. Descripción

El plugin es.upv.dsic.issi.moment.intergenomics.cpn.edit proporciona la funcionalidad básica para representar los elementos del modelo de *Cpn Tools* en un editor.

6.1.5.2. Dependencias

Las dependencias de este plugin son:

- org.eclipse.core.runtime (por ser un plugin de Eclipse)
- org.eclipse.emf.edit (por ser un plugin de EMF-Edit)
- es.upv.dsic.issi.moment.intergenomics.cpn (por ser el soporte para edición del metamodelo de *CPN Tools*).

6.1.5.3. Descripción de paquetes y clases

El contenido de este plugin se ha generado de forma automática, aunque se han realizado cambios menores en el código (principalmente en los métodos getText() de los diferentes *ItemProviders*).

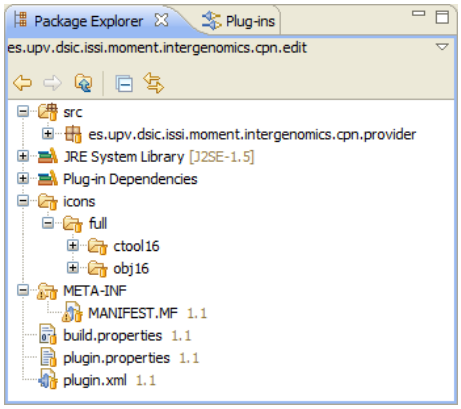


Figura 6.9: Proyecto es.upv.dsic.issi.moment.intergenomics.cpn.edit.

Paquete es.upv.dsic.issi.moment.intergenomics.cpn.provider

Este es el único paquete del plugin. Contiene la clase activadora del plugin (CpnEditPlugin), que se encarga de controlar el cicli de vida del plugin mediante su instancia *singleton*. A su vez, contiene la clase CpnItemProviderAdapterFactory que es la clase de la factoría para acceder a cada uno de los *items providers* contenidos en el paquete.

Por último, contiene una clase *provider* por cada clase del modelo (todas con el sufijo *–ItemProvider*). Estas clases sirven para consultar cómo se deben mostrar los elementos en los editores, esto es, qué icono representará a cada elemento, cómo se construirá el texto de sus etiquetas, qué elementos del modelo son sus *hijos* (de manera que se presentan los correspondientes menús de creación de *nuevo hijo*), qué propiedades deben de mostrarse en las hojas de propiedades para poder ser visualizadas/editadas, etc.

Listado de clases y métodos

A continuación se listan las clases del paquete. Además, se listan los métodos de la clase AnnotItemProvider como ejemplo de los métodos proporcionados por un *ItemProvider*.

- AnnotItemProvider.java
  - AnnotItemProvider(AdapterFactory) – Constructor
  - getPropertyDescriptors(Object)
  - addTextPropertyDescriptor(Object)[1]
  - addIdPropertyDescriptor(Object)
  - getImage(Object)
  - getText(Object)
  - notifyChanged(Notification)
  - collectNewChildDescriptors(Collection<Object>, Object)
  - getResourceLocator()
- ArcItemProvider.java
- AuxBoxItemProvider.java
- AuxEllipseItemProvider.java
- AuxiliaryItemProvider.java
- AuxTextItemProvider.java
- BlockItemProvider.java
- ColorSetElementItemProvider.java
- ColorSetItemProvider.java
- CompoundColorSetItemProvider.java

[1]Los diferentes métodos *–PropertyDescriptor* dependen de las propiedades de cada clase del modelo.

- CondItemProvider.java
- CpnEditPlugin.java
- CpnetItemProvider.java
- CpnItemProviderAdapterFactory.java
- DeclarationItemProvider.java
- DiagramElementItemProvider.java
- EnumeratedItemProvider.java
- FusionItemProvider.java
- GlobboxItemProvider.java
- GlobrefItemProvider.java
- GroupItemProvider.java
- InitmarkItemProvider.java
- MarkItemProvider.java
- MlItemProvider.java
- PageItemProvider.java
- PlaceItemProvider.java
- PortItemProvider.java
- ProductItemProvider.java
- SimpleColorSetItemProvider.java
- TransItemProvider.java
- VarItemProvider.java

6.1.6. Plugin es.upv.dsic.issi.moment.intergenomics.cpn.editor

6.1.6.1. Descripción

Este plugin implementa un editor en árbol típico de EMF. Se encuentra asociado por defecto con la extensión «*.cpn». Además, incluye el asistente de creación de un nuevo fichero «*.cpn» integrado con el resto de asistentes de creación de recursos de Eclipse.

6.1.6.2. Dependencias

El plugin es.upv.dsic.issi.moment.intergenomics.cpn.editor tiene las siguientes dependencias:

- org.eclipse.core.runtime
- org.eclipse.core.resources
- org.eclipse.ui.ide
- org.eclipse.emf.ecore.xmi
- org.eclipse.emf.edit.ui
- es.upv.dsic.issi.moment.intergenomics.cpn.edit

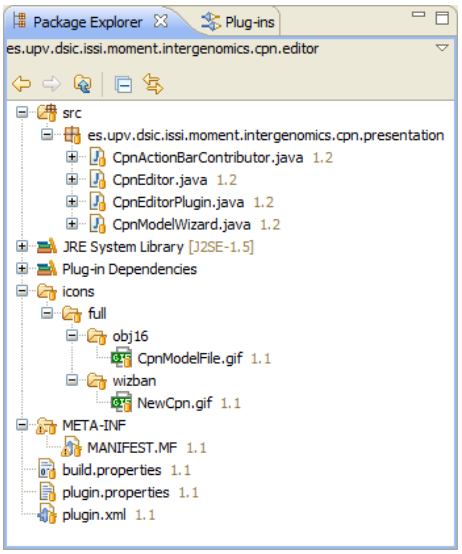


Figura 6.10: Proyecto es.upv.dsic.issi.moment.intergenomics.cpn.editor.

6.1.6.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.moment.intergenomics.cpn.presentation

Este es el único paquete del plugin y su código es completamente generado de forma automática. Contiene a su vez las siguiente clases:

CpnActionBarContributor

La clase CpnActionBarContributor contiene las acciones que se contribuirán a la interfaz estándar de Eclipse. Como contribuciones entendemos los botones que se añadirán a la barra de herramientas de Eclipse, el menú que aparecerá en la barra de menús cuando el editor se encuentra abierto y activo, etc.

CpnEditor

La clase CpnEditor implementa el editor en árbol para las instancias del modelo de *CPN Tools*. En cuanto a funcionalidad es similar a cualquier otro editor en árbol genérico, siendo su única diferencia en cómo en la inicialización del editor (método initializeEditingDomain()) se añade el *ItemProviderAdapterFactory* (creado en el plugin es.upv.dsic.issi.moment.intergenomics.cpn.edit comentado anteriormente) al adapterFactory propio del editor.

```
1       //...
2       adapterFactory = new ComposedAdapterFactory(ComposedAdapterFactory.Descriptor.
            Registry.INSTANCE);
```

```
3
4       adapterFactory.addAdapterFactory(new ResourceItemProviderAdapterFactory());
5       adapterFactory.addAdapterFactory(new CpnItemProviderAdapterFactory());
6       adapterFactory.addAdapterFactory(new ReflectiveItemProviderAdapterFactory());
7       //...
```

Listado 6.8: Fragmento de código del método initializeEditingDomain() de la clase CpnEditor.

CpnEditorPlugin

La clase CpnEditorPlugin es la clase activadora del plugin conteniendo la instancia *singleton* que permite acceder al estado de éste y controlar su ciclo de vida.

CpnModelWizard

Esta clase implementa el asistente de creación de una nueva instancia del modelo de *CPN Tools*. El asistente se integra en la categoría «Example EMF Model Creation Wizards».

6.2. Preprocesado y postprocesado

6.2.1. Plugin es.upv.dsic.issi.moment.intergenomics.bridges

6.2.1.1. Descripción

El plugin es.upv.dsic.issi.moment.intergenomics.bridges es el principal plugin de las tareas de pre- y post-procesado de los datos. En él se implementa la lógica fundamental que permite reconstruir una instancia en XMI del metamodelo de TRANSPATH®a partir de un fichero XML extraído de la base de datos así como los mecanismos de proyección a un XML de *CPN Tools* a partir de un XMI que conforma al metamodelo EMF de *CPN Tools*.

6.2.1.2. Dependencias

Como plugin de Eclipse, este plugin depende de:

- org.eclipse.ui
- org.eclipse.core.runtime
- org.eclipse.core.resources – Permite acceder a los recursos del *workspace*.
- org.eclipse.ui.console – Permite volcar mensajes en la vista de consola de Eclipse.

- es.upv.dsic.issi.moment.intergenomics.transpath – Permite tratar con instancias del metamodelo de TRANSPATH®.
- es.upv.dsic.issi.moment.intergenomics.cpn – Permite tratar con instancias del modelo de *CPN Tools*.

Además, importa e incluye las siguientes bibliotecas:

- log4j-1.2.14.jar. Esta biblioteca proporciona una API extendida para gestionar mensajes de *logging*.
- jdom.jar. Esta biblioteca proporciona una API para navegar y tratar con documentos XML.

6.2.1.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.moment.intergenomics.bridges

Este paquete únicamente contiene la clase IntergenomicsBridgesPlugin, que es la clase activadora del plugin. Ésta se encarga de controlar el ciclo de vida del plugin (métodos start() y stop()), así como proporcionar los métodos de acceso a la funcionalidad del plugin. La clase contiene a su vez dos variables estáticas transpathParser y cpnetProjector, instancias de las clases *TranspathParser* y *CpnToolsProjector*, que comentaremos más adelante.

```
1       private static TranspathParser transpathParser = new TranspathParser();
2       private static CpnToolsProjector cpnetProjector = new CpnToolsProjector();
```

Listado 6.9: Declaración de las variables transpathParser y cpnetProjector

Los métodos que se han añadido a la implementación por defecto son:

createNetwork(IFile) Este método accede a los contenidos de un *IFile* (archivo del workspace) e invoca al método createNetwork(InputStream) para que realice la reconstrucción de la red de Petri.

```
1       public Network createNetwork(IFile file) throws InitialiseException,
            CoreException {
2         return createNetwork(file.getContents());
3       }
```

Listado 6.10: Método createNetwork(IFile)

createNetwork(InputStream) Este método toma un stream de texto, e invoca al método createNetwork(InputStream) de la clase *TranspathParser*. El valor devuelto es la red de Petri, instancia del metamodelo de EMF de *CPN Tools*, que representa a la red.

```
1       public Network createNetwork(InputStream stream) throws InitialiseException,
            CoreException {
2         return transpathParser.createNetwork(stream);
3       }
```

Listado 6.11: Método createNetwork(InputStream)

saveAndLayoutCpnet(Cpnet, IFile, IProgressMonitor) Este método invoca al proyecto de código implementado en el método saveCpnetForCpnTools(Cpnet cpnet, IFile file, boolean reLayout, IProgressMonitor monitor) de la clase *CpnToolsProjector*. Nótese que en este caso, el flag reLayout está activo, forzando que antes del paso de proyección, se llame al método performLayout() de la clase *PageImpl*.

```
1       public void saveAndLayoutCpnet(Cpnet cpnet, IFile file, IProgressMonitor
            monitor) throws ParserConfigurationException, IOException, CoreException {
2         cpnetProjector.saveCpnetForCpnTools(cpnet, file, true, monitor);
3       }
```

Listado 6.12: Método saveAndLayoutCpnet(Cpnet, IFile, IProgressMonitor)

saveCpnet(Cpnet, IFile, IProgressMonitor) Este método invoca al proyecto de código implementado en el método saveCpnetForCpnTools(...) de la clase *CpnToolsProjector*. En este caso, el flag reLayout no está activo, proyectándose en el fichero XML final los elementos en la misma posición que la indicada en el documento en XMI.

```
1       public void saveCpnet(Cpnet cpnet, IFile file, IProgressMonitor monitor) throws
            ParserConfigurationException, IOException, CoreException {
2         cpnetProjector.saveCpnetForCpnTools(cpnet, file, false, monitor);
3       }
```

Listado 6.13: Método saveCpnet(Cpnet, IFile, IProgressMonitor)

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.parser

Este paquete contiene las clases InitialiseException y TranspathParser. La primera de ellas, es una excepción que será lanzada en caso de que ocurra algún error al intentar parsear el XML de TRANSPATH®(el fichero XML no esté accesible, sea un fichero XML mal formado, etc.).

La clase TranspathParser por su parte, implementa el parser de ficheros XML. Se trata una adaptación al *framework* EMF del trabajo expuesto en [76], y hace uso de las librerías JDOM [56] y log4j [23] para parsear el documento, y mostrar información al usuario. La mayor parte de la lógica de esta clase está implementada en el método createNetwork(InputStream). El código de esta adaptación se encuentra en el Anexo I.

```
1       public Network createNetwork(InputStream stream) throws InitialiseException,
            CoreException;
```

Listado 6.14: Signatura del método createNetwork(...)

Paquete es.upv.dsic.issi.moment.intergenomics.cpntools.projector

El paquete es.upv.dsic.issi.moment.intergenomics.cpntools.projector contiene la única clase *CpnToolsProjector*, y en ella se implementa todo el mecanismo de proyección de código a la herramienta de *CPN Tools*.

Como constantes propias de la clase, contiene una serie de valores por defecto, que se aplicarán a la hora de definir el tamaño de los distintos elementos gráficos, así como para establecer los valores constantes del documento XML.

El método a través del que se accederá a la funcionalidad de la clase es saveCpnetForCpnTools(...) (listado 6.15). En este método se creará la instancia de la clase *org.w3c.dom.Document* (línea 13), que se corresponderá con el documento XML que se irá construyendo. Tras crear el documento, se inicializa la raíz del documento, indicando la herramienta, versión, y demás atributos específicos de *CPN Tools* (15–23).

El siguiente paso es invocar al método createCpnet(document,cpnet) (25). Esta invocación inicia el proceso de proyección, lanzando un proceso en cascada, recorriendo el modelo EMF de cpntools hacia abajo, e invocando al resto de métodos que implementa esta clase. Cuando se alcanza la línea 29 del método, el proceso de proyección ya se ha completado, teniendo en la variable document el fichero XML final que debe ser salvado. En las líneas 29–45 se configura el formato de persistencia final, y se realiza el volcado en un archivo en disco.

```
1       public void saveCpnetForCpnTools(Cpnet cpnet, IFile file, boolean reLayout,
            IProgressMonitor monitor)
2         throws ParserConfigurationException, IOException, CoreException {
3
4         BINDER_POS_X = new Float(cpnet.getPage().getPosx());
5         BINDER_POS_Y = new Float(cpnet.getPage().getPosy());
6         BINDER_WIDTH = new Float(cpnet.getPage().getWidth());
7         BINDER_HEIGHT = new Float(cpnet.getPage().getHeight());
8
9         if (reLayout) {
10          cpnet.getPage().performLayout(new Float(BINDER_WIDTH*0.9).intValue(), new Float(
                BINDER_HEIGHT*0.9).intValue(), 5000);
11        }
12
13        Document document = DocumentBuilderFactory.newInstance().newDocumentBuilder().
              newDocument();
14
15        Element root = document.createElement("workspaceElements");
16        document.appendChild(root);
17
```

```
18          Element generatorNode = document.createElement("generator");
19          generatorNode.setAttribute("tool", "CPN Tools");
20          generatorNode.setAttribute("version", "2.0.0");
21          generatorNode.setAttribute("format", "5");
22
23          root.appendChild(generatorNode);
24
25          root.appendChild(createCpnet(document,cpnet));
26
27          // Saving the result
28
29          OutputFormat format = new OutputFormat(document);
30          format.setEncoding("iso-8859-1");
31          format.setDoctype("-//CPN//DTD CPNXML 1.0//EN", "http://www.daimi.au.dk/~cpntools/
                bin/DTD/5/cpn.dtd");
32            format.setIndenting(true);
33            format.setIndent(2);
34
35            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
36
37            XMLSerializer serializer = new XMLSerializer(outputStream, format);
38            serializer.serialize(document);
39
40            InputStream inputStream = new ByteArrayInputStream(outputStream.toByteArray());
41
42          if (file.exists())
43            file.setContents(inputStream, IResource.KEEP_HISTORY, monitor);
44          else
45            file.create(inputStream, IResource.KEEP_HISTORY, monitor);
46
47        }
```

Listado 6.15: Implementación del método saveCpnetForCpnTools(...)

Para ilustrar como se realiza el proceso de proyección de código, mostraremos el método createCpnet(...).

```
1       private Node createCpnet(Document document, Cpnet cpnet) {
2
3           Element element = document.createElement("cpnet");
4
5           element.appendChild(createGlobbox(document, cpnet.getGlobbox()));
6           element.appendChild(createPage(document, cpnet.getPage()));
7           Object obj = new Object();
8           element.appendChild(createInstances(document, cpnet.getPage(), getModelElementId(
                obj)));
9           element.appendChild(createBinders(document, getModelElementId(obj)));
10          return element;
11      }
```

Listado 6.16: Método createCpnet(...)

Como se observa en el listado 6.16, en primer lugar, se crea un elemento (instancia de

la clase *org.w3c.dom.Element*), que se corresponde con un *tag* de el documento XML. Este tag será el string «cpnet». en este punto, y dado el elemento cpnet del modelo, se consultan los elementos que cuelgan de él directamente (cpnet.getGlobbox(), cpnet.getPage(), etc.) y se invoca a los correspondiente métodos de creación (createGlobbox(...), createPage(...), etc.). Cada uno de los métodos createXxxxx(...) está definido de forma análoga, de esta manera, se va recorriendo en profundidad el árbol del modelo, hasta construir el documento final completo. El conjunto de métodos que implementa la clase *CpnToolsProjector* es:

- CpnToolsProjector() – Constructor
- createAnnot(Document, Annot)
- createArc(Document, Arc)
- createBinders(Document, String)
- createBlock(Document, Block)
- createBox(Document, Float, Float)
- createColorSet(Document, ColorSet)
- createCpnet(Document, Cpnet)
- createElementId(Document, String)
- createEllipse(Document, Float, Float)
- createEnumerated(Document, Enumerated)
- createGlobbox(Document, Globbox)
- createInitmark(Document, Initmark)
- createInstances(Document, Page, String)
- createLayoutForEnumeratedColorset(Document, Enumerated)
- createLayoutForProductColorset(Document, Product)
- createPage(Document, Page)
- createPlace(Document, Place)
- createPlaceType(Document, Place)
- createProduct(Document, Product)
- createText(Document, String)
- createTransition(Document, Trans)
- fillElementAttributesFromDiagramElement(Document, Element, DiagramElement)
- getModelElementId(Object)
- saveCpnetForCpnTools(Cpnet, IFile, boolean, IProgressMonitor)

6.2.2. Plugin es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui

6.2.2.1. Descripción

El plugin es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui implementa los elementos que se contribuyen a la interfaz gráfica de Eclipse para poder invocar de forma manual a los puentes de interoperabilidad (pre– y postprocesado) implementados en el plugin es.upv.dsic.issi.moment.intergenomics.bridges. Específicamente, este plugin implementa un menú contextual llamado «Intergenomics» que aparecerá en el «Explorador de proyectos» de Eclipse, y permitirá invocar diversas accionas sobre un determinado fichero que esté seleccionado. Para ello, se define una clase que implementa la interfaz *org.eclipse.ui.IObjectActionDelegate* por cada opción del menú. Posteriormente, en el archivo de manifiesto «plugin.xml» se define la conexión al punto de extensión para que la acción se contribuya a la interfaz.

En el listado 6.17 se muestra un ejemplo de cómo la clase es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui.popup.actions.ParseFile se conecta al punto de extensión de los menús popup de Eclipse.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <?eclipse version="3.2"?>
3   <plugin>
4      <extension
5            point="org.eclipse.ui.popupMenus">
6         <objectContribution
7               objectClass="org.eclipse.core.resources.IFile"
8               nameFilter="*.*"
9               id="es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui.
                    contribution1">
10            <menu
11                  label="Intergenomics"
12                  path="additions"
13                  id="es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui.menu1">
14               <separator
15                     name="group1">
16               </separator>
17            </menu>
18            <action
19                  label="Parse Transpath XML file"
20                  class="es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui.popup
                        .actions.ParseFile"
21                  menubarPath="es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui
                        .menu1/group1"
22                  enablesFor="1"
23                  id="es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui.popup.
                        actions.ParseFile">
24            </action>
25         </objectContribution>
26      </extension>
```

```
27      </plugin>
```

Listado 6.17: Ejemplo de conexión a un punto de extensión de un menú popup

6.2.2.2. Dependencias

El plugin tiene las siguientes dependencias:

- org.eclipse.ui – Proporciona el entorno de ejecución del entorno gráfico básico de Eclipse.
- org.eclipse.ui.ide – Proporciona acceso a algunas clases fundamentales del entorno, como cuadros de diálogo predefinidos.
- org.eclipse.core.runtime – Proporciona el soporte básico de la plataforma.
- org.eclipse.core.resources – Permite tratar con los recursos del *workspace*.
- es.upv.dsic.issi.moment.intergenomics.transpath – Permite tratar con instancias del metamodelo de TRANSPATH®.
- es.upv.dsic.issi.moment.intergenomics.cpn – Permite tratar con instancias del metamodelo de *CPN Tools*.
- es.upv.dsic.issi.moment.intergenomics.bridges – Proporciona la lógica para parsear y proyectar los documentos XML nativos.

6.2.2.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui

El paquete es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui únicamente contiene la clase activadora del plugin, *TranspathParserUIPlugin*, que controla el ciclo del vida del plugin. En este caso, el código de esta clase ha sido generado de forma automática y contiene la implementación estándar.

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.parser.ui.popup.actions

Este paquete contiene las clases que implementan las acciones que seran ejecutadas al seleccionar las distintas opciones. Todas implementan la interfaz *org.eclipse.ui.IObject-ActionDelegate*, implementando de forma similar los métodos run() y selectionChanged() (hereadados de *org.eclipse.ui.IActionDelegate*).

Método selectionChanged(...)

El método selectionChanged(...) es invocado cada vez que la selección de recursos del *workspace* varía y se invoca una determinada acción. Es en este método donde se deben recuperar los datos que interesen de la selección, y almacenarla en los atributos de clase para poder ser accedidos posteriormente en el método run(...).

Una implementación típica de este método (que es común a las tres clases del paquete) se muestra en el listado 6.18. En esta implementación se consulta el primer elemento de la selección (el menú contextual sólo estará disponible en caso de que se seleccione un único elemento), y se almacena en la variable file en caso de que la selección sea un fichero del *workspace* (instancia de *IFile*).

```
1  public void selectionChanged(IAction action, ISelection selection) {
2    file = null;
3    if(selection instanceof IStructuredSelection) {
4      IStructuredSelection sel = (IStructuredSelection)selection;
5      Object selElem = sel.getFirstElement();
6      if(selElem instanceof IFile)
7        file = (IFile)selElem;
8    }
9  }
```

Listado 6.18: Método selectionChanged(...)

Método ParseFile.run(...)

Este es el método que invocará al parser de TRANSPATH®de forma manual. Fundamentalmente, llamará al parser pasando como argumento el fichero seleccionado, creará un nuevo recurso de EMF, al que le añadirá el resultado del proceso anterior, y salvará el recurso en disco, tal y como muestra el fragmento de código mostrado en 6.19.

```
1  public void run(IAction action) {
2    //...
3    try {
4      Network network = IntergenomicsBridgesPlugin.getDefault().createNetwork(file);
5
6      Resource resource = new ResourceSetImpl().createResource(URI.createFileURI(file.
           getFullPath().removeFileExtension().addFileExtension("xmi").toPortableString
           ()));
7
8      resource.getContents().add(network);
9
10     resource.save(Collections.EMPTY_MAP);
11
12   } catch (InitialiseException e) {
13   //...
14   } catch (CoreException e) {
15   //...
16   } catch (IOException e) {
```

```
17          //...
18          }
19        }
```

Listado 6.19: Método ParseFile.run(...)

Métodos SaveAndLayoutNetworkFile.run(...) y saveNetworkFile.run(...)

La implementación de los métodos SaveAndLayoutNetworkFile.run(...) y saveNetworkFile.run(...) son similares, estribando la única diferencia en si al invocar al plugin del proyector de código, se invoca al método IntergenomicsBridgesPlugin.getDefault().saveAndLayoutCpnet(...) o al método IntergenomicsBridgesPlugin.getDefault().saveCpnet(...). En el listado 6.20 se muestra un ejemplo de la implementación.

En primer lugar, se carga en memoria el fichero XMI con la red de Petri a transformar. Posteriormente, se solicita el nombre del fichero de destino, y finalmente se invoca al proyector de código pasando como argumento la red de Petri a procesar, y el fichero destino. La clase *UIJob* únicamente se emplea para poder ejecutar este trabajo utilizando la API de trabajos y monitores de la interfaz de Eclipse.

```
1       public void run(IAction action) {
2         Shell shell = new Shell();
3         try {
4
5           final Resource resource = new ResourceSetImpl().createResource(URI.createFileURI(
                file.getFullPath().toPortableString()));
6
7           resource.load(Collections.EMPTY_MAP);
8
9           new UIJob(shell.getDisplay(), "Save as XML") {
10
11            @Override
12            public IStatus runInUIThread(IProgressMonitor monitor) {
13
14              monitor.beginTask(null, IProgressMonitor.UNKNOWN);
15
16              IFile resultFile = ResourcesPlugin.getWorkspace().getRoot().getFile(
17                  file.getFullPath().removeFileExtension().addFileExtension("xml"));
18
19              SaveAsDialog saveAsDialog = new SaveAsDialog(getDisplay().getActiveShell());
20
21              saveAsDialog.setTitle("Save as XML");
22              saveAsDialog.setOriginalFile(resultFile);
23
24              if (saveAsDialog.open() == Dialog.OK) {
25                resultFile = ResourcesPlugin.getWorkspace().getRoot().getFile(saveAsDialog.
                      getResult());
26              }
27
28              try {
```

```
29                IntergenomicsBridgesPlugin.getDefault().saveAndLayoutCpnet((Cpnet) resource
                      .getContents().get(0),
30                    resultFile,monitor);
31              } catch (ParserConfigurationException e) {
32              //...
33              } catch (IOException e) {
34              //...
35              } catch (CoreException e) {
36              //...
37              }
38              return Status.OK_STATUS;
39            }}.schedule();
40
41        } catch (IOException e) {
42              //...
43        }
44      }
```

Listado 6.20: Métodos SaveAnLayoutNetworkFile.run(...) y saveNetworkFile.run(...)

6.2.3. Plugin es.upv.dsic.issi.moment.xmlemf

6.2.3.1. Descripción

El plugin es.upv.dsic.issi.moment.xmlemf implementa una serie de clases auxiliares y de utilidades que permiten extender la plataforma Eclipse para permitir que el *framework* de EMF trate de forma transparente con los ficheros XML extraídos de TRANSPATH®. De esta manera, el proceso de *parseo* del fichero se realiza de forma automática y al vuelo cada vez que se intenta cargar un recurso que validad el DTD de la base de datos de TRANSPATH®indicando su URI. Esto permite que para tratar con estos ficheros XML no sea necesario realizar el preproceso del fichero de forma manual empleando los mecanismos implementados en el plugin indicado en la sección 6.2.2.

6.2.3.2. Dependencias

Este plugin depende de:

- org.eclipse.core.runtime
- org.eclipse.emf.ecore
- org.eclipse.core.resources
- org.eclipse.emf.ecore.xmi
- es.upv.dsic.issi.moment.intergenomics.transpath
- es.upv.dsic.issi.moment.intergenomics.bridges

6.2.3.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.moment.intergenomics.transpath.xmlemf

Éste es el único paquete que contiene el plugin, y contiene las clases TranspathXMLPlugin, TranspathXMLContentType, TranspathXMLResourceFactoryImpl y TranspathXMLResourceImpl.

Clase TranspathXMLPlugin

La primera de ellas se corresponde con la habitual clase activadora, empleada para controlar el ciclo de vida del plugin. En este caso, la clase TranspathXMLPlugin contiene una implementación habitual sin métodos personalizados.

Clase TranspathXMLContentType

La clase *TranspathXMLContentType* implementa la interfaz org.eclipse.core.runtime.content.ITextContentDescriber e implementa la lógica para decidir si el contenido de un fichero XML es un fichero XML válido según el DTD de TRANSPATH®de forma que Eclipse pueda discernir con qué tipo de editor se debe abrir este tipo de fichero (dado que una extensión «*.xml» es excesivamente genérica para poder describir con exactitud el tipo de contenido).

Esta clase fundamentalmente implementa el método describe(InputStream , IContentDescription) devolviendo IContentDescriber.VALID si el *stream* representa un texto válido y IContentDescriber.INVALID si no lo es. Se considera un texto válido si al invocar al plugin del parser éste tiene éxito (listado 6.21).

```
1       public int describe(InputStream inputstream,
2           IContentDescription icontentdescription) throws IOException {
3         try {
4           IntergenomicsBridgesPlugin.getDefault().createNetwork(inputstream);
5         } catch (InitialiseException e) {
6           return IContentDescriber.INVALID;
7         } catch (CoreException e) {
8           return IContentDescriber.INVALID;
9         }
10        return IContentDescriber.VALID;
11      }
```

Listado 6.21: Método describe(...)

Mediante la conexión al punto de extensión mostrado en 6.22 se muestra cómo se le indica a Eclipse que se debe emplear esta clase.

```
1       <extension point="org.eclipse.core.runtime.contentTypes">
2         <content-type
3           base-type="org.eclipse.core.runtime.xml"
```

```
4           file-extensions="xml,tp"
5           id="es.upv.dsic.issi.moment.intergenomics.transpath.editor.XMLcontentTypeBinding"
6           name="Transpath XML"
7           priority="high">
8           <describer
9             class="es.upv.dsic.issi.moment.intergenomics.transpath.xmlemf.
                  TranspathXMLContentType">
10          </describer>
11        </content-type>
12      </extension>
```

Listado 6.22: Uso del *extension point* org.eclipse.core.runtime.contentTypes

Clase TranspathXMLResourceFactoryImpl

La clase *TranspathXMLResourceFactoryImpl* implementa una factoría para los recursos de tipo *modelo de TRANSPATH®*. De esta manera, mediante el punto de extensión org.eclipse.emf.ecore.extension_parser se puede indicar a EMF que los ficheros con una cierta extensión (en este caso «*.xml» y «*.tp») pueden cargarse mediante una determinada factoría (listado 6.23).

```
1       <extension
2         point="org.eclipse.emf.ecore.extension_parser">
3         <parser
4           class="es.upv.dsic.issi.moment.intergenomics.transpath.xmlemf.
                  TranspathXMLResourceFactoryImpl"
5           type="xml">
6         </parser>
7         <parser
8           class="es.upv.dsic.issi.moment.intergenomics.transpath.xmlemf.
                  TranspathXMLResourceFactoryImpl"
9           type="tp">
10        </parser>
11      </extension>
```

Listado 6.23: Conexión al punto de extensión

La factoría (listado 6.24) únicamente debe implementar el método createResource(...), que devuelve un objeto que ha de ser de tipo *Resource* (interfaz org.eclipse.emf.ecore.resource.Resource).

```
1       public class TranspathXMLResourceFactoryImpl extends ResourceFactoryImpl{
2
3         public Resource createResource(URI uri) {
4           return new TranspathXMLResourceImpl(uri);
5         }
6       }
```

Listado 6.24: Clase TranspathXMLResourceFactoryImpl

Clase TranspathXMLResourceImpl

La clase *TranspathXMLResourceImpl* implementa la interfaz *Resource* y extiende la clase *ReourceImpl*, ambas de la API de EMF. Este tipo de *recurso* de EMF permite encapsular modelos ya parseados desde la base de datos de TRANSPATH®. Para ello, se deben implementar los métodos doSave(...) y doLoad(...), indicando cómo se debe de construir el artefacto EMF en memoria a la hora de cargar el recurso; y cómo se debe persistir el recurso en disco al querer salvarlo.

En el caso de *TranspathXMLResourceImpl*, el método doLoad(...) invoca al parser de TRANSPATH®pasándole como argumento el archivo al que se corresponde el *recurso*. En cuanto al método doSave(...), no se permite salvar los cambios realizados sobre el modelo en memoria (ya que no se ha implementado el mecanismo de proyección de un XML de TRANSPATH®). Por ello, lo que se realiza es una copia exacta sobre el stream del fichero resultado de los contenidos del fichero XML original, lanzándose una advertencia al usuario (a través del registro de errores de Eclipse) indicando que el guardado de este tipo de recursos no está permitido.

6.3. Transformaciones en QVT-Relations

6.3.1. MOMENT

6.3.1.1. Regla NetworkToCpnet

La regla inicial de la transformación crea, tal como muestra la regla *NetworkToCpnet* (Fig. 6.11), un elemento *Cpnet*, un elemento *Page* y un elemento *Globbox* para cada *Network* de la instancia origen. El elemento *Cpnet* es el elemento raíz del modelo destino. Por su parte, la clase *Page* es la que contendrá a todos los elementos gráficos del modelo resultante, y la clase *Globbox* la que contendrá las declaraciones de datos (*ColorSets*). De esta manera, tras crear estos elementos iniciales, mediante el bloque `where` se lanza la creación de los bloques de declaraciones (regla *ComplexMoleculeToComplexesBlock*) así como de los elementos visuales (regla *ReactionToGUIElements*).

6.3.1.2. Regla ComplexMoleculeToComplexesBlock

La regla *ComplexMoleculeToComplexesBlock* (Fig. 6.12) especifica que, si existe un *globbox* en el modelo destino y una *molécula* compleja en el modelo origen en el momento de aplicación de la regla, se deberá crear un *bloque* —*Block*— que estará contenido en el *globbox* que ya existe, cuyo nombre identificativo (*idname*) será `'Complexes'`. Puesto que *idname* es el atributo clave de la clase *Block*, en sucesivas aplicaciones de la regla no se crearán *bloques* adicionales. El bloque `where` además indica que se debe cumplir como post-condición de esta regla la regla *ComplexMoleculeToProduct*.

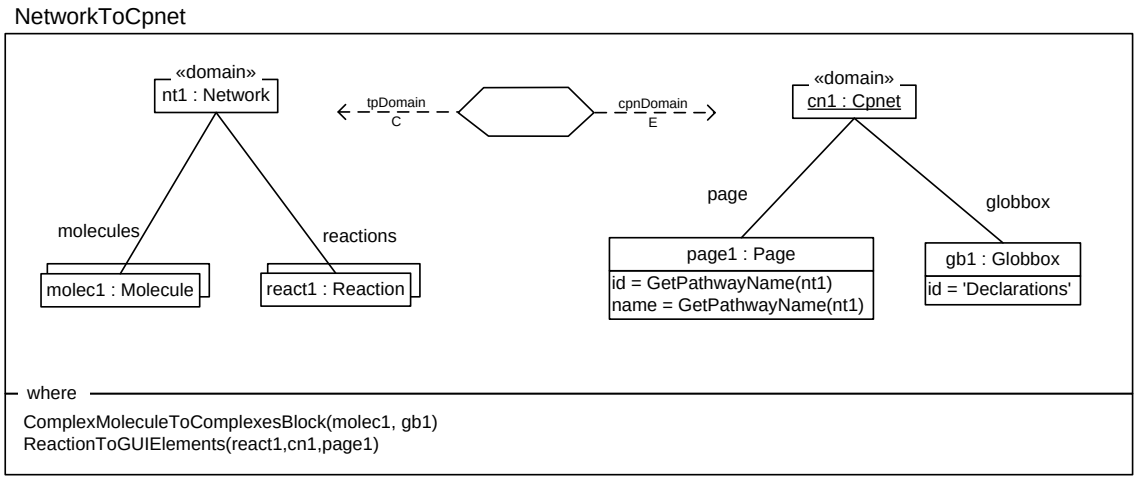


Figura 6.11: Regla *NetworkToCpnet* en MOMENT–QVT.

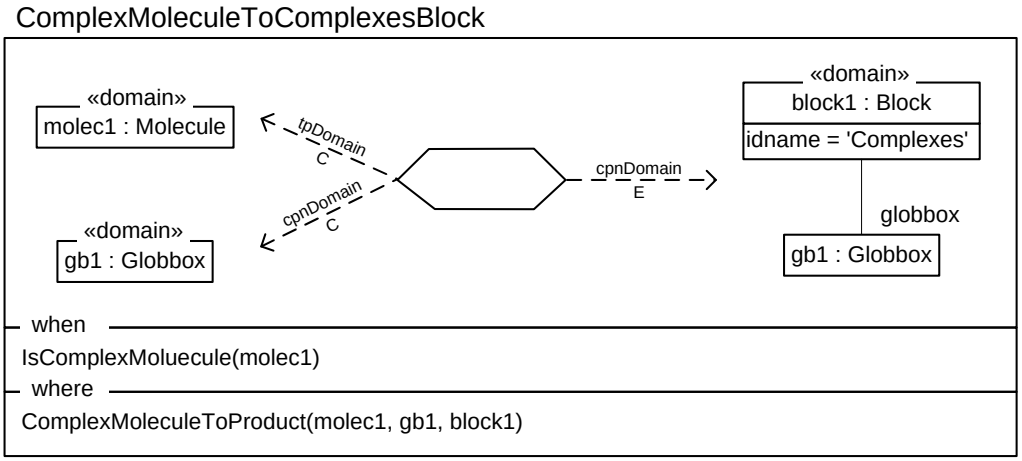


Figura 6.12: Regla *ComplexMoleculeToComplexesBlock* en MOMENT–QVT.

```
1       function IsComplexMolecule(molec:Molecule):Bool
2       {
3           (not(molec.statesOf -> isEmpty()))
4       }
```

Listado 6.25: Helper IsComplexMolecule(molec:Molecule).

6.3.1.3. Regla ComplexMoleculeToProduct

La figura 6.13 comprueba que si existe al menos una *molécula* está compueta por otras más simples (rol *statesOf*), un *globbox* y un *block* (*block1*), debe existir en el modelo destino un elemento de tipo *Product* contenido en el bloque *block1* y cuyo atributo *idname*

sea igual al resultado de la aplicación de la consulta OCL encapsulada en la función *GetMoleculeType(...)* (listado 6.26). El bloque `where` indica las postcondiciones de la forma habitual.

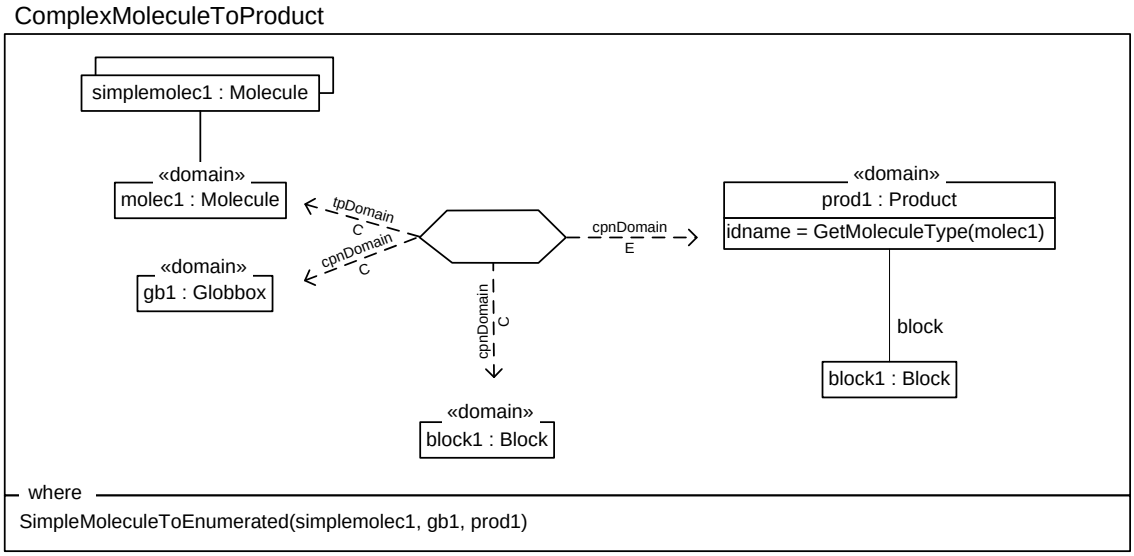


Figura 6.13: Regla *ComplexMoleculeToProduct* en MOMENT–QVT.

```
1       function GetMoleculeType(molec : Molecule) : String
2       {
3           if (molec.statesOf -> isEmpty())
4           then
5              if (molec.klass -> includes('adaptor proteins'))
6               then 'A'
7               else
8                   if (molec.klass -> includes('receptors'))
9                   then 'R'
10                  else 'O'
11                  endif
12              endif
13          else
14              if (molec.name = 'LPS:LBP')
15              then 'OA'
16              else
17                  if (molec.name = 'LPS:LBP:CD14')
18                  then 'OAR'
19                  else
20                      if
21                      (molec.name = 'ST2:TIRAP' or molec.name = 'ST2:MyD88')
22                      then 'RA'
23                      else 'Unknown' // No debería llegar aquí
24                      endif
25                  endif
26              endif
```

```
27            endif
28        }
```

Listado 6.26: Helper GetMoleculeType(molec : Molecule).

6.3.1.4. Regla SimpleMoleculeToEnumerated

La regla *SimpleMoleculeToEnumerated* (figura 6.14) especifica que, si en el modelo origen existen al menos un *globbox* (*gb1*), un *product* (*prod1*), y una *molécula* (*molec1*) (donde además ésta es de tipo simple —se cumple la cláusula `when` *IsSimpleMolecule(molec1)*—), deberá existir en el modelo destino un elemento de tipo *Enumerated*. Este elemento, llamado *enum1*, debe usarse en el producto *prod1* (rol *usedIn*), y debe estar contenido en un *bloque* de nombre `'Ressources'` dentro del *globbox gb1*. Por último, especifica que el nombre identificativo de *enum1* debe coincidir con el resultado devuelto de la expresión OCL *GetMoleculeType* indicada anteriormente (listado 6.26). La postcondición especificada en la cláusula `where` será la que se encargará de poblar (si no existen ya) los elementos simples que se encuentran dentro de *prod1*.

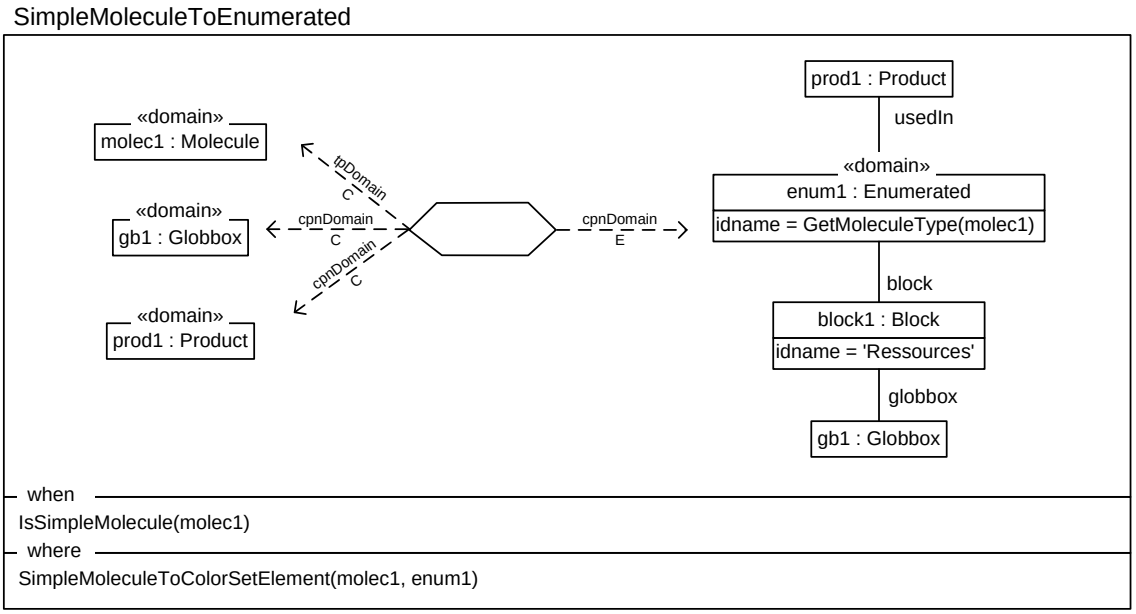


Figura 6.14: Regla *SimpleMoleculeToEnumerated* en MOMENT–QVT.

6.3.1.5. Regla SimpleMoleculeToColorSetElement

La regla *SimpleMoleculeToColorSetElement* (fig. 6.15) comprueba que en el momento de la aplicación de la regla exista al menos en el modelo origen una *molécula*, y en el modelo destino un elemento de tipo *Enumerated* (*enum1*). Con esto, fuerza a su vez que

en el modelo destino debe existir un elemento de tipo *ColorSetElement* que esté contenida en *enum1* y cuyo nombre sea el mismo que el de la molécula del modelo origen con la que se ha hecho *binding*. Esta regla no tiene ninguna pre- ni postcondición.

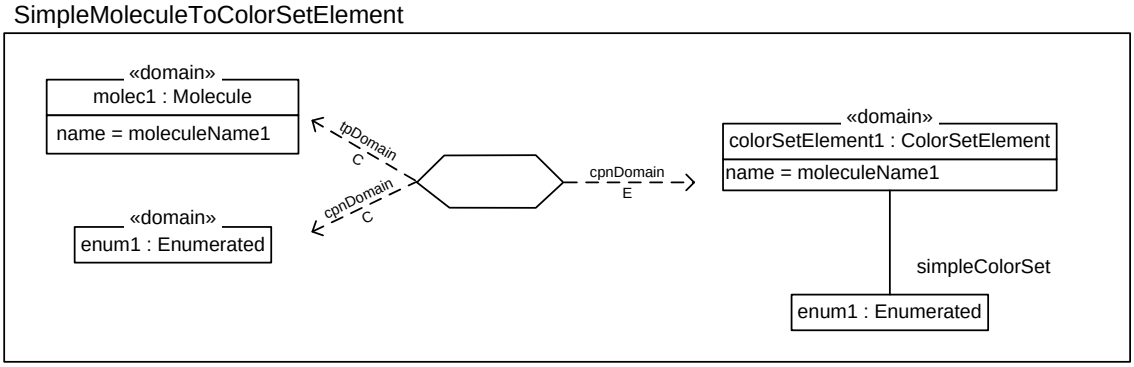


Figura 6.15: Regla *SimpleMoleculeToColorSetElement* en MOMENT–QVT.

6.3.1.6. Regla ReactionToGUIElements

La regla *ReactionToGUIElements* (fig. 6.16) es la primera que se ejecuta para poblar el modelo destino con los elementos gráficos que deben mostrarse en la red de Petri final. En este sentido, en primer lugar comprueba que debemos tener en el modelo origen al menos una reacción, y ésta debe tener un nombre, una serie de reactivos (a los que se llegará mediante la navegación del rol *reactantsCoefficient*) y una serie de productos (a los que se llegará mediante el rol *producesCoefficient*). Además, en el modelo destino debe existir una red de Petri (elemento *cn1*, de tipo *Cpnet*) y una página (*page1*). Tras esta comprobaciones, la regla debe asegurar que para cada reacción existente en el modelo origen, debe existir un elemento de tipo *Trans* cuyo nombre debe ser el mismo que el nombre de la reacción, cuyo texto debe set `'bind'` y que debe estar contenido en la página *page1*. El *binding* con las variables *rc1* y *pc1* se empleará respectivamente para propagarlo en la comprobación de las postcondiciones *ReactantsdCoefficientToPlaces* y *ProductCoefficientToPlace*, que son las reglas que crearán los elementos del modelo destino tanto para los reactivos como los productos de la reacción seleccionada al aplicar la regla.

6.3.1.7. Regla ReactantsCoefficientToPlaces

La regla *ReactantsCoefficientToPlaces* es trivial, y únicamente se emplea para navegar el rol *reactants* del coeficiente (*ReactantCoefficient*) *rc1*. A su vez, comprueba que existen un elemento *Cpnet*, *Page* y *Trans*. La postcondición de la regla invoca a *ReactantsToPlaces*, que recibe la variable *reactantMolecule1*, que es el resultado de la navegación del rol *reactants*.

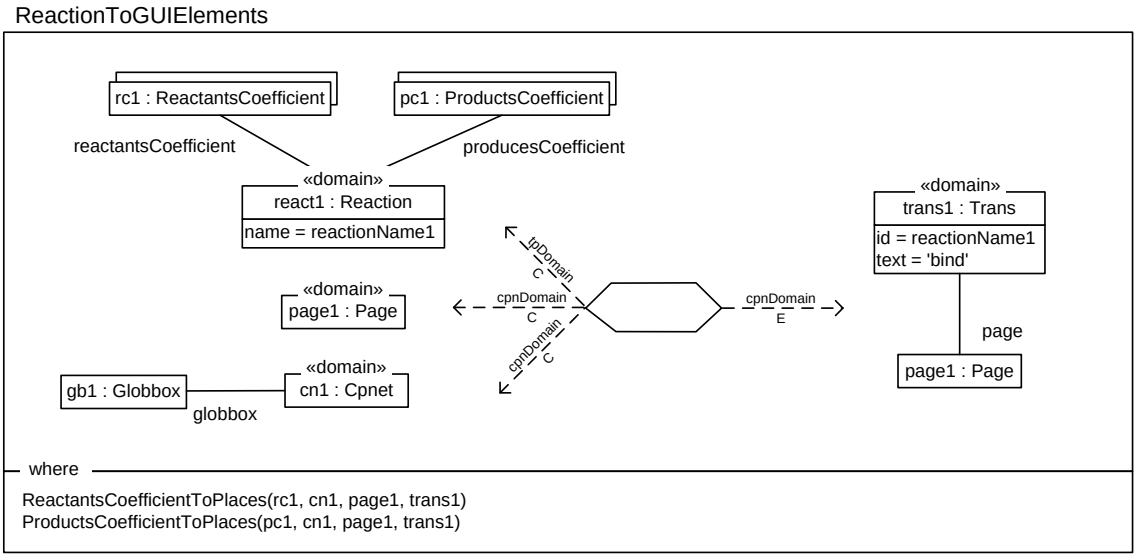


Figura 6.16: Regla *ReactionToGUIElements* en MOMENT–QVT.

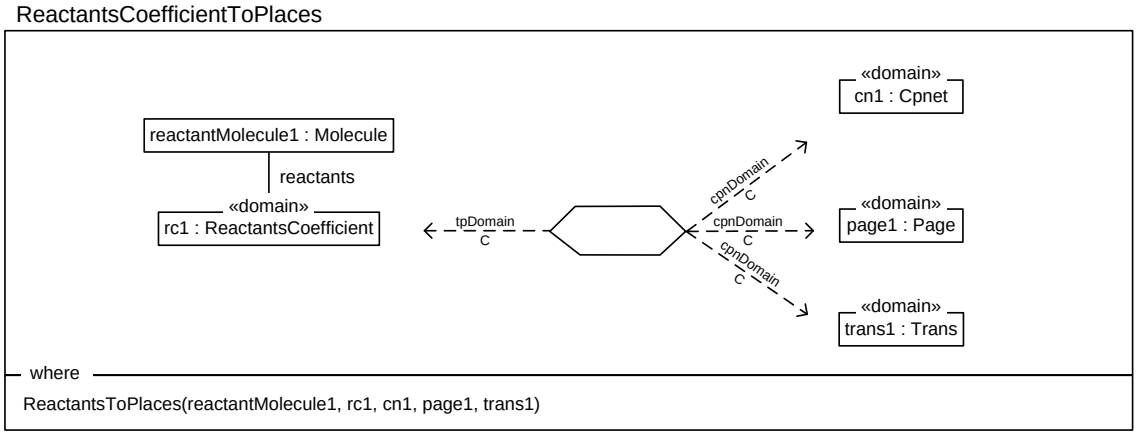


Figura 6.17: Regla *ReactantsCoefficientToPlaces* en MOMENT–QVT.

6.3.1.8. Regla ReactantsToPlaces

En la relación *ReactantsToPlaces* se pretende crear un elemento de tipo *Place* por cada uno de los reactivos que intervienen en una *reacción*. Para ello, se comprueba que existe una *molécula* que hace *binding* con la variable *reactantMolecule1* y que debe de tener un nombre. A su vez, en el modelo destino deben existir elementos de tipo *Cpnet*, *Page* y *Trans* (con un identificador). Con estos *bindings* establecidos, debe forzarse que se cumpla que en el modelo destino debe existir un elemento de tipo *Place*, contenido en la *página*, cuyo identificador se corresponda con el tipo de la molécula (*GetMoleculeType(reactantMolecule1)*), y esté referenciado por un elemento de tipo *Fusion* (contenido en

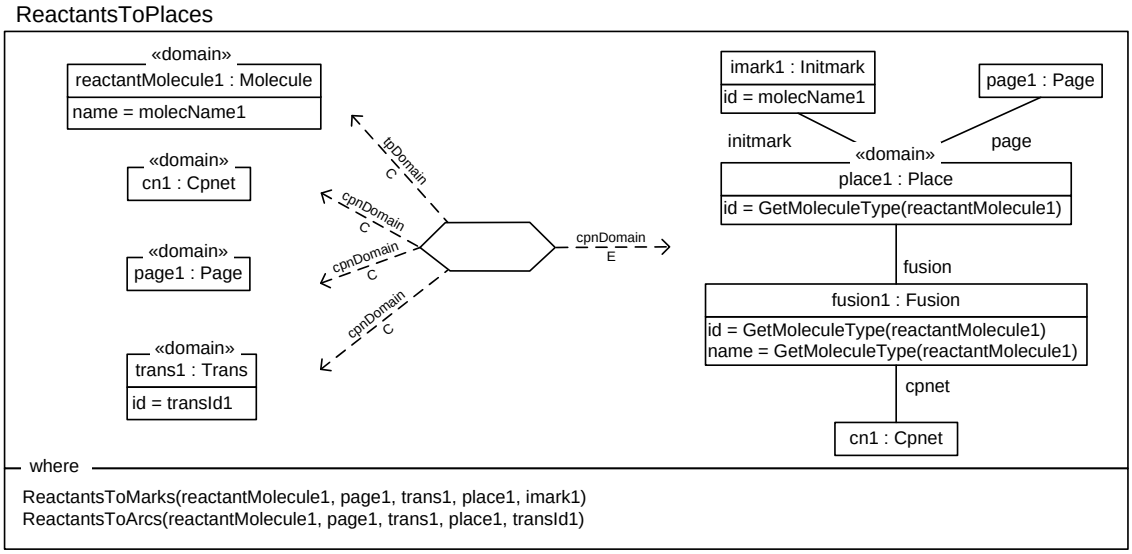


Figura 6.18: Regla *ReactantsToPlaces* en MOMENT–QVT.

la *Cpnet*) y cuyo nombre e identificador también coinciden con el tipo de molécula. Por último, este elemento de tipo *Fusion* debe contener un elemento de tipo *Initmark* (el token con el que está marcado inicialmente el place), cuyo nombre coincida con el de la molécula dada.

La postcondición *ReactantstoMarks* se encargará posteriormente de establecer el número de tokens iniciales del *Place*, y la regla *ReactantsToArcs*, establecerá el vínculo (arco) entre el *Place* reción creado y el elemento de tipo *Trans* con el que se relaciona.

6.3.1.9. Regla ReactantsToMarks

En la relación *ReactantsToPlaces* se pretende crear un elemento de tipo *Place* por cada uno de los reactivos que intervienen en una *reacción*. Para ello, se comprueba que existe una *molécula* que hace *binding* con la variable *reactantMolecule1* y que debe de tener un nombre. A su vez, en el modelo destino deben existir elementos de tipo *Cpnet*, *Page* y *Trans* (con un identificador). Con estos *bindings* establecidos, debe forzarse que se cumpla que en el modelo destino debe existir un elemento de tipo *Place*, contenido en la *página*, cuyo identificador se corresponda con el tipo de la molécula (*GetMoleculeType(reactantMolecule1)*), y esté referenciado por un elemento de tipo *Fusion* (contenido en la *Cpnet*) y cuyo nombre e identificador también coinciden con el tipo de molécula. Por último, este elemento de tipo *Fusion* debe contener un elemento de tipo *Initmark* (el token con el que está marcado inicialmente el place), cuyo nombre coincida con el de la molécula dada.

La postcondición *ReactantstoMarks* se encargará posteriormente de establecer el nú-

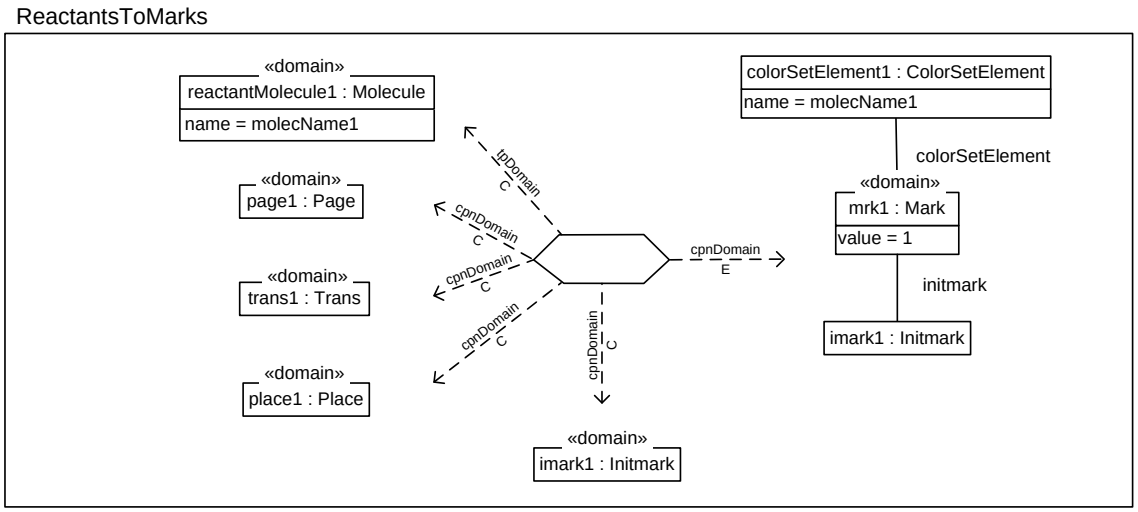


Figura 6.19: Regla *ReactantsToMarks* en MOMENT–QVT.

mero de tokens iniciales del *Place*, y la regla *ReactantsToArcs*, establecerá el vínculo (arco) entre el *Place* reción creado y el elemento de tipo *Trans* con el que se relaciona.

6.3.1.10. Regla ProductsCoefficientToPlaces

La relación *ProductsCoefficientToPlaces* 6.20 es trivial, y únicamente se emplea para navegar el rol *produces* del coeficiente (*ProductsCoefficient*) *pc1*. A su vez, comprueba que existen un elemento *Cpnet*, *Page* y *Trans*. La postcondición de la regla invoca a *Products-ToPlaces*, que recibe la variable *productMolecule1*, que es el resultado de la navegación del rol *produces*.

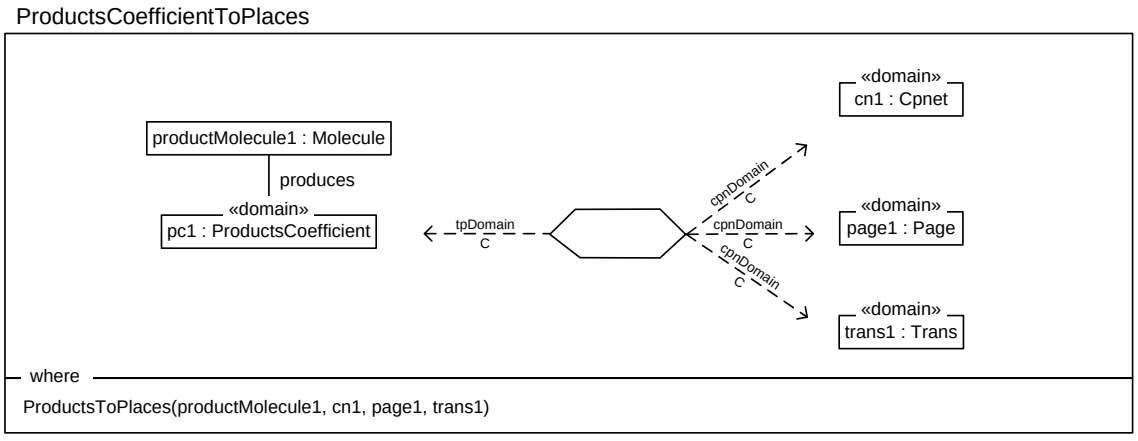


Figura 6.20: Regla *ProductsCoefficientToPlaces* en MOMENT–QVT.

6.3.1.11. Regla ProductsToPlaces

En la relación *ProductsToPlaces* 6.21 se pretende crear un elemento de tipo *Place* por cada uno de los productos que intervienen en una *reacción*. Para ello, se comprueba que existe una *molécula* que hace *binding* con la variable *productMolecule1*. A su vez, en el modelo destino deben existir elementos de tipo *Cpnet*, *Page* y *Trans* (con un identificador). Con estos *bindings* establecidos, debe forzarse que se cumpla que en el modelo destino debe existir un elemento de tipo *Place*, contenido en la *página*, cuyo identificador se corresponda con el tipo de la molécula (*GetMoleculeType(reactantMolecule1)*), y esté referenciado por un elemento de tipo *Fusion* (contenido en la *Cpnet*) y cuyo nombre e identificador también coinciden con el tipo de molécula.

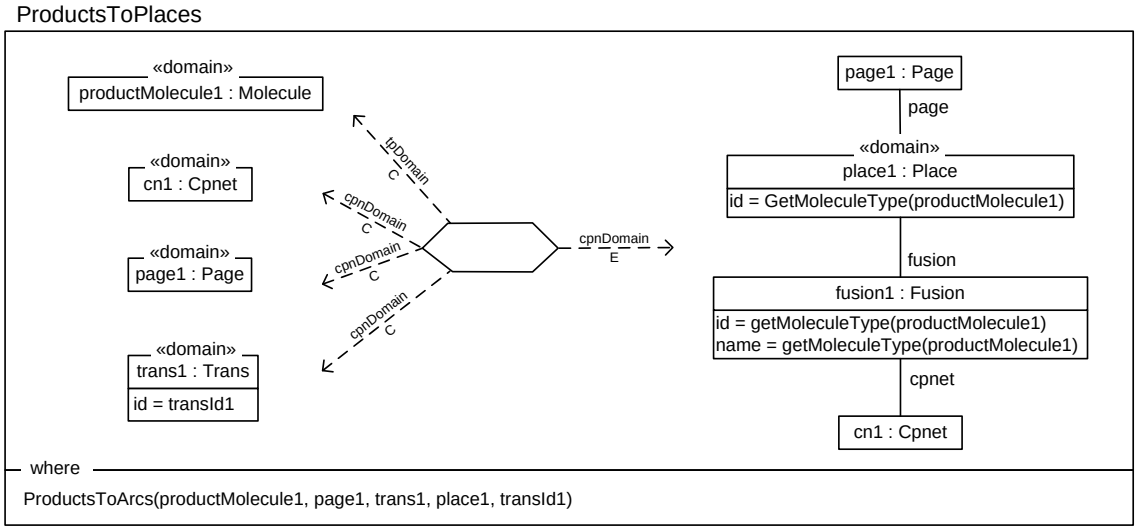


Figura 6.21: Regla *ProductsToPlaces* en MOMENT–QVT.

La postcondición *ReactantstoArcs* se encargará posteriormente de establecer el vínculo (arco) entre el *Place* reción creado y el elemento de tipo *Trans* con el que se relaciona.

6.3.1.12. Regla ReactantsToArcs

La relación *ReactantsToArcs* 6.22 comprueba que deben existir en el momento de invocación de la regla al menos una molécula con nombre (*reactantMolecName1*), una *página*, un elemento de tipo *Trans*, un elemento de tipo *Place* y un elemento de tipo *String* (dominio primitivo *transId1*, que se corresponde con el identificador de un elemento de tipo *Trans*). En caso de que todas estas condiciones se cumplan, se debe forzar en el dominio destino que debe existir un elemento de tipo *Arc* (un arco) contenido en la *página*, y cuyo identificador se corresponda con la expresión "`'{' + reactantMolecName1 + '} =>{' + transId1 + '}'`". Este elemento *Arc* debe ser de tipo `'PtoT'` (atributo *orientation*), y enlazará los elementos *Trans* y *Place* que hacen *binding* con las variables `trans1` y `place1`

de la regla.

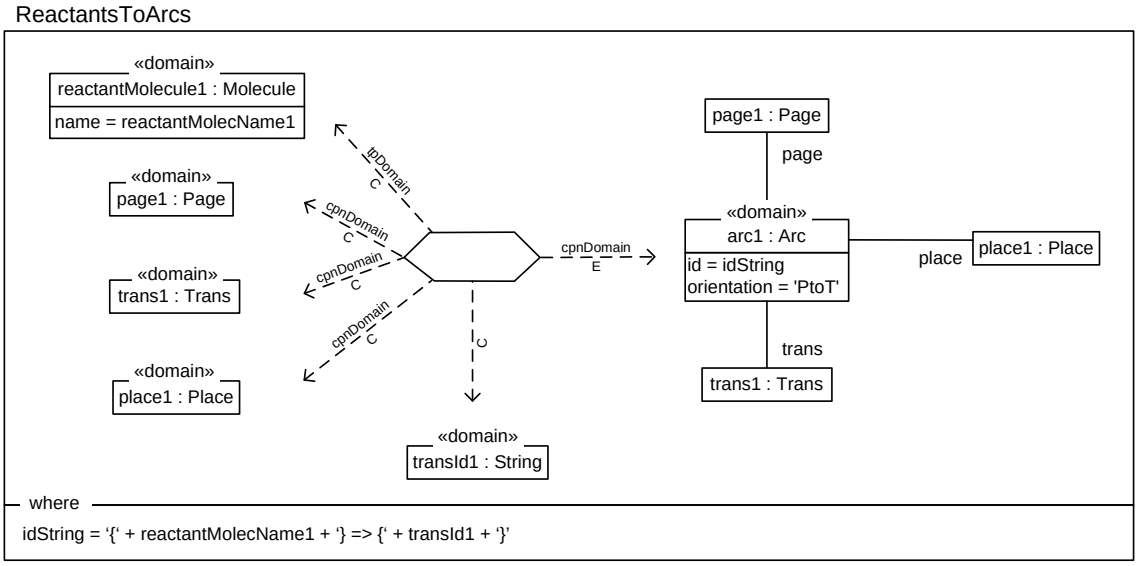


Figura 6.22: Regla *ReactantsToArcs* en MOMENT–QVT.

6.3.1.13. Regla ProductsToArcs

La relación *ProductsToArcs* 6.23 es muy similar a la regla anterior *ReactantsToArcs* 6.22. En este caso, comprueba que deben existir en el momento de invocación de la regla al menos una molécula con nombre (*productMolecName1*), una *página*, un elemento de tipo *Trans*, un elemento de tipo *Place* y un elemento de tipo *String* (dominio primitivo *transId1*). En caso de que todas estas condiciones se cumplan, se debe forzar en el dominio destino que debe existir un elemento de tipo *Arc* (un arco) contenido en la *página*, y cuyo identificador se corresponda con la expresión "`'{' + transId1 + '} =>{' + reactantMolecName1 + '}'`". Este elemento *Arc* debe ser de tipo `'TtoP'` —TransToPlace— (atributo *orientation*), y enlazará los elementos *Trans* y *Place* que hacen *binding* con las variables `trans1` y `place1` de la regla.

6.3.2. Medini QVT

6.3.2.1. Regla NetworkToCpnet

La regla *NetworkToCpnet* es la regla inicial de la transformación, y crea el elemento raíz del modelo destino (esto es, una instancia de la clase *Cpnet*). A su vez, crea cada una de las instancias que englobarán los elementos gráficos (el elemento de tipo *Page*) y las declaraciones (el elemento de tipo *Globbox*). La página se crea por defecto en la posición (200, 100), para evitar que se solape con la barra de declaraciones izquierda.

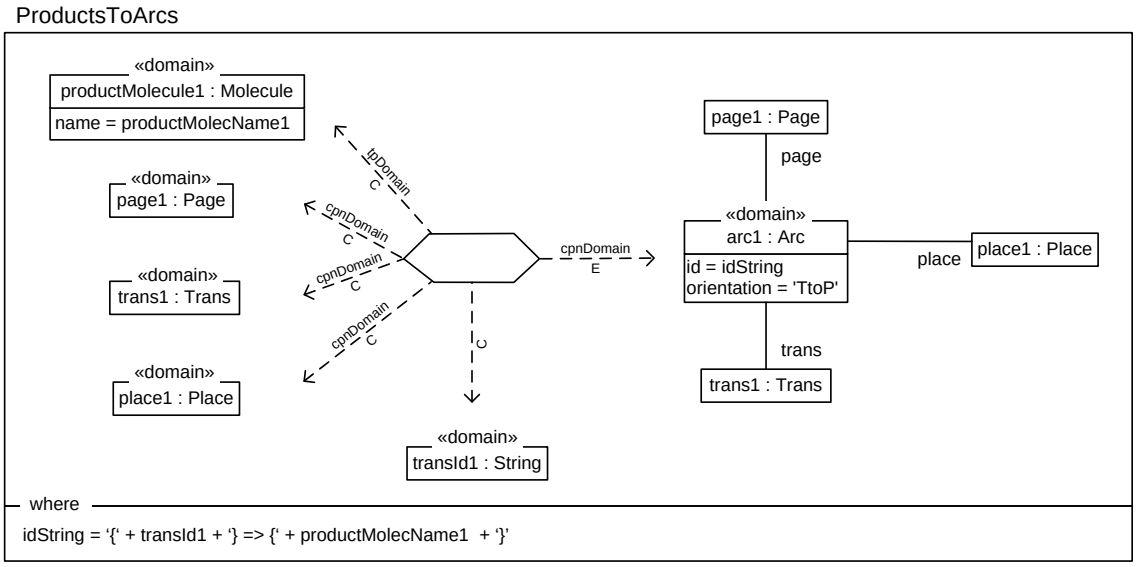


Figura 6.23: Regla *ProductsToArcs* en MOMENT–QVT.

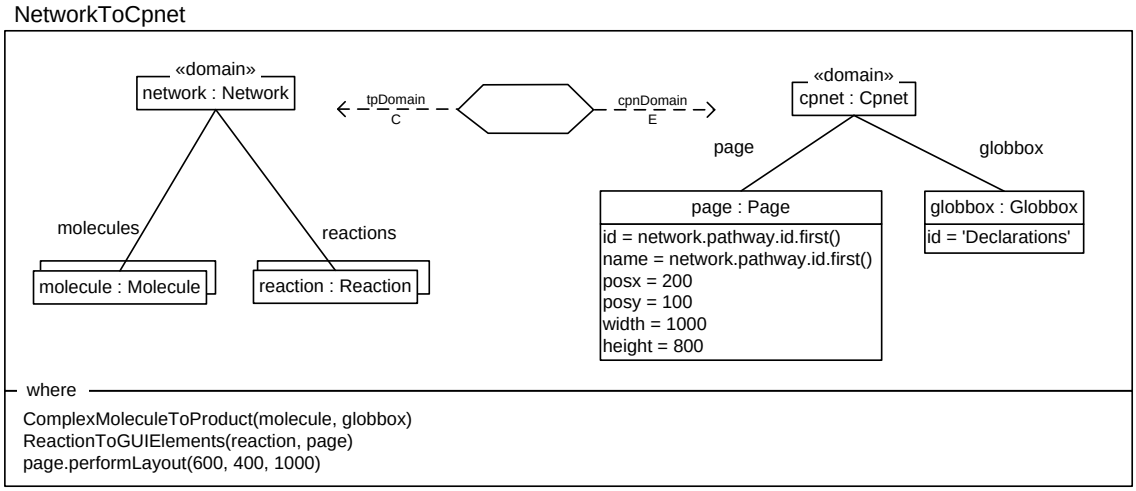


Figura 6.24: Regla *NetworkToCpnet* en MediniQVT.

En las postcondiciones se puede observar que se ejecuta el método page.performLayout(...), esto permite ejecutar el algoritmo de redibujado dentro de la propia transformación. Como otras postcondiciones se encuentran la aplicación de las reglas *ComplexMoleculeToProduct* y *ReactionToGUIElements*. La primera de ellas lanza la generación de la parte de declaraciones, a partir del conjunto de moléculas y la segunda lanza la generación de los elementos gráficos a partir del conjunto de reacciones.

6.3.2.2. Regla ComplexMoleculeToProduct

La regla *ComplexMoleculeToProduct* 6.25 genera un producto por cada molécula compleja que hace *binding* con el dominio `checkonly` de la relación. Adicionalmente, en la parte `enforce` del dominio obliga a que existan (y si no existen los creará) sendos bloques de nombres «`Resources`» y «`Complexes`». El elemento de tipo *Product* estará agrupado en el grupo *complexeBlock*.

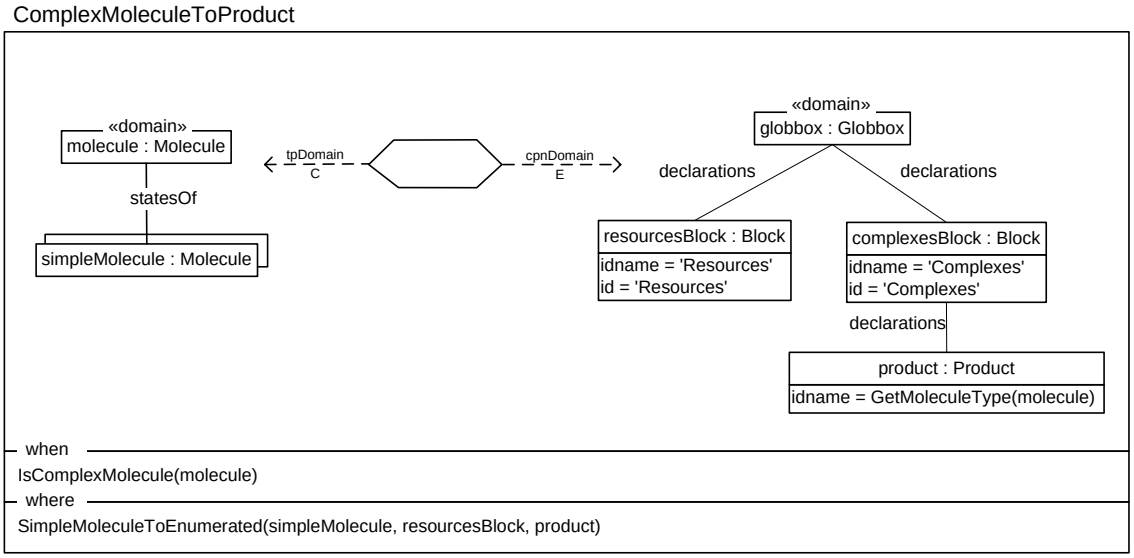


Figura 6.25: Regla *ComplexMoleculeToProduct* en MediniQVT.

La regla *ComplexMoleculeToProduct* únicamente se ejecutará en caso de que la molécula que hace *binding* con ella sea una molécula compleja 6.27.

```
1       query IsComplexMolecule(molec:Molecule):Boolean
2       {
3         (not(molec.statesOf -> isEmpty()))
4       }
```

Listado 6.27: Query IsComplexMolecule(...)

Como postcondición se ejecutará la regla *SimpleMoleculeToEnumerated*, invocándose para cada una de las moléculas simples que forman la molécula compleja.

6.3.2.3. Regla SimpleMoleculeToEnumerated

La regla *SimpleMoleculeToEnumerated* 6.26 se ejecuta haciendo binding con las moléculas de tipo simple (según indica la postcondición 6.28).

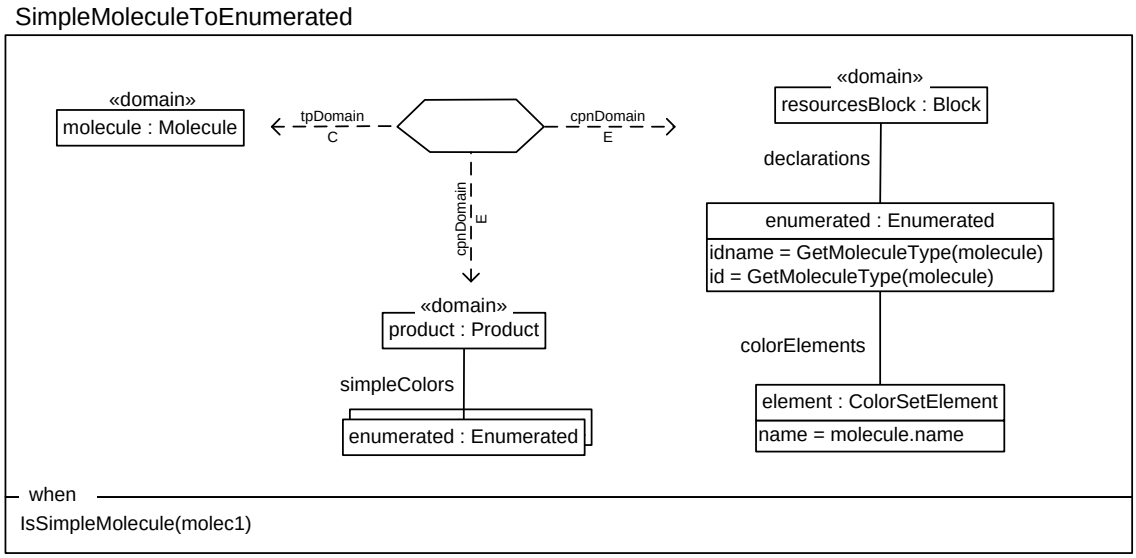


Figura 6.26: Regla *SimpleMoleculeToEnumerated* en MediniQVT.

```
1   query IsSimpleMolecule(molec:Molecule):Boolean
2   {
3     (molec.statesOf -> isEmpty())
4   }
```

Listado 6.28: Query IsSimpleMolecule(...)

Para cada una de ellas se creará un *ColorSetElement*, cuyo nombre será el de la molécula. Este *ColorSetElement* formará parte de un elemento de tipo *Enumerated* que se corresponderá con el *Enumerated* del tipo de la molécula 6.29. Este *Enumerated*, por su parte, formará parte del *product* que se debe haber generado para la molécula compleja de la que la molécula simple forma parte.

```
1   query GetMoleculeType(molecule : transpath::Molecule) : String
2   {
3     if IsSimpleMolecule(molecule) then
4       GetSimpleMoleculeType(molecule)
5     else
6       GetComplexMoleculeType(molecule)
7     endif
8   }
9
10  query GetSimpleMoleculeType(molec : Molecule) : String
11  {
12    if (molec.klass -> includes('adaptor proteins'))
13    then 'A'
14    else
15      if (molec.klass -> includes('receptors'))
16      then 'R'
```

```
17          else '0'
18          endif
19        endif
20      }
21
22      query GetComplexMoleculeType(complexMolecule : Molecule) : String
23      {
24        complexMolecule.statesOf
25        ->collect(m : Molecule | GetSimpleMoleculeType(m))
26        ->asSet()
27        ->sortedBy(type : String | type )
28        ->iterate(type : String; complexBlockID : String = '' | complexBlockID.concat(type)
              )
29      }
```

Listado 6.29: Query GetMoleculeType(...)

6.3.2.4. Regla ReactionToGUIElements

La regla *ReactionToGUIElements* se ejecuta para cada una de las reacciones del modelo origen. Para cada uno de ellas crea un elemento de tipo *Trans* con los atributos que aparecen en la figura 6.27 que estará contenido en el elemento de tipo *Page* del modelo. El elemento *Trans* se identificará por el nombre de la reacción, y el texto que contendrá se construye empleando la consulta OCL *BuildTransitionText(...)* 6.30.

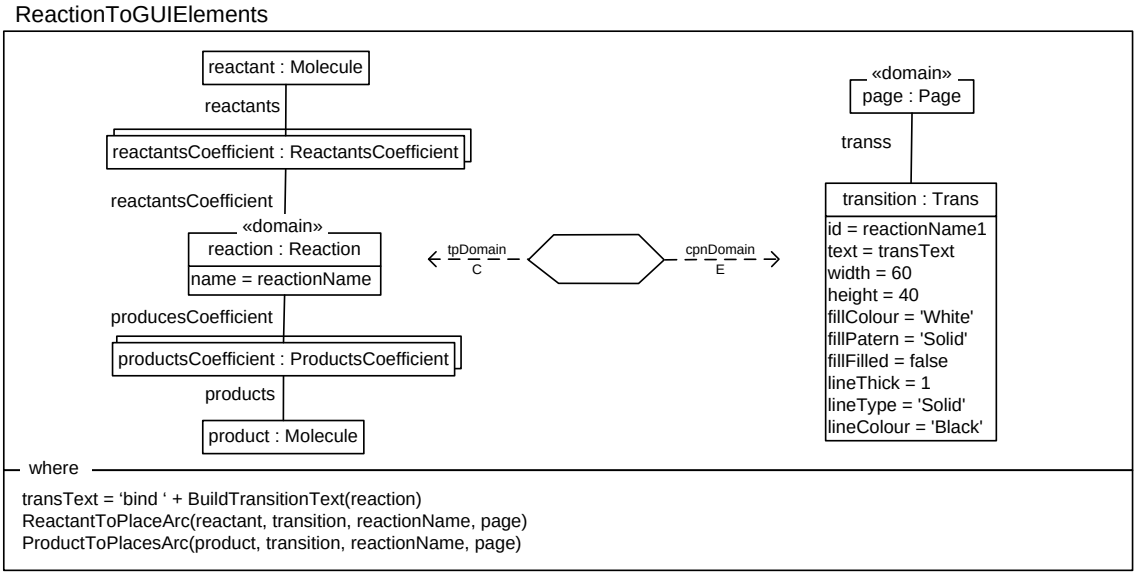


Figura 6.27: Regla *ReactionToGUIElements* en MediniQVT.

Las postcondiciones *ReactantToPlaceArc* y *ProductToPlaceArc* lanzan la creación de los correspondientes *Places* (y sus *arcos*) que irán enlazados con este elemento de tipo

Trans. La postcondición *ReactantToPlaceArc* se ejecutará por cada una de las moléculas reactantes de la reacción, y la postcondición *ProductRoPlaceArc* se ejecutará por cada uno de los productos.

```
1       query BuildTransitionText(reaction : Reaction) : String
2       {
3         let moleculeNames : Sequence(String) =
4           reaction.reactantsCoefficient.reactants.name
5         in
6           moleculeNames
7           ->excluding(moleculeNames.last())
8           ->iterate(moleculeName : String; text : String = '' | text.concat(moleculeName).
                concat(','))
9           .concat(moleculeNames.last())
10      }
```

Listado 6.30: Query BuildTransitionText(...)

6.3.2.5. Regla ReactantToPlaceArc

La regla *ReactantToPlaceArc* 6.28 crea para una molécula del dominio origen (que será un reactante de una reacción), y un elemento de tipo *Trans* del dominio destino, el correspondiente *place* que representa a la molécula en la red de Petri, así como el arco

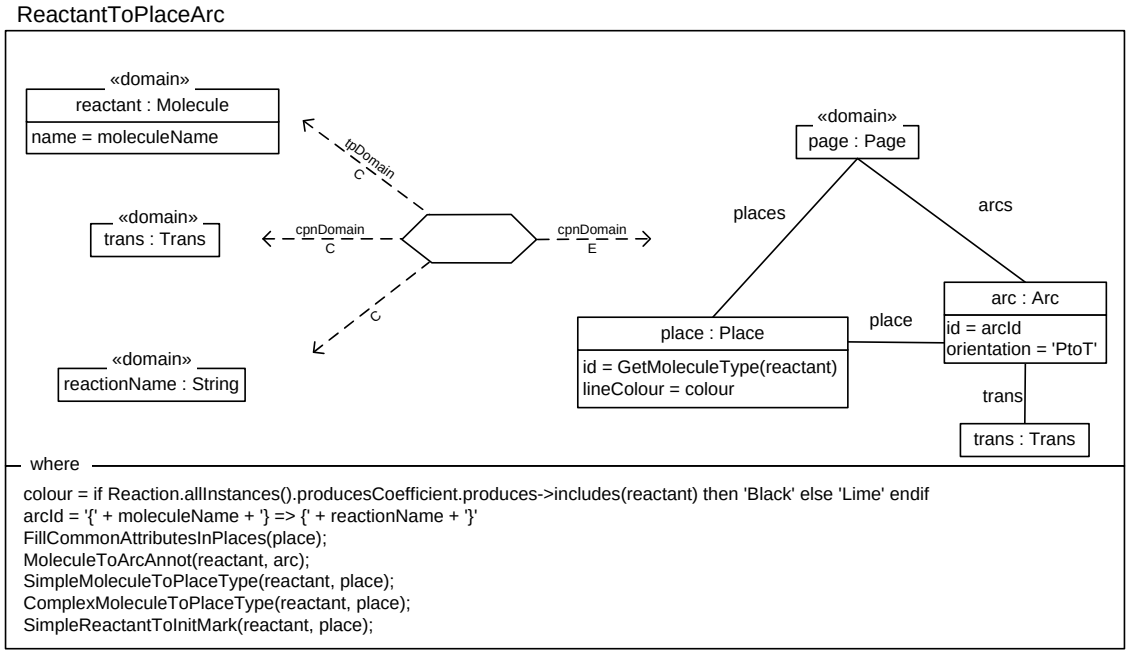


Figura 6.28: Regla *ReactantToPlaceArc* en MediniQVT.

que que relaciona el elemento de tipo *Place* con el elemento de tipo *Trans*. Además, dichos elemento que se crean deben estar contenidos en un elemento de tipo *Page*.

Como identificador del *place* se emplea el tipo de ma molécula, extraido de la consulta OCL *GetMoleculeType(. . .)* 6.29.

El conjunto de poscondiciones que se ejecutan completarán la información que aquí se establece. La regla *FillCommonAttributesInPlaces* establece los attributos típicos de un *place*, para evitar duplicar información en diferentes reglas. Las reglas *SimpleMoleculeToPlaceType* y *ComplexMoleculeToPlaceType* establece el tipo del *place* dependiendo del tipo de molécula y las reclas *MoleculeToArcAnnot* y *SimpleReactantToInitMark* establecen el texto de los arcos y las marcas iniciales de los *places* respectivamente.

6.3.2.6. Regla ProductToPlaceArc

La regla *ReactantToPlaceArc* 6.29 es similar a la regla anterior. Ésta crea para una molécula del dominio origen (que será un producto de una reacción), y un elemento de tipo *Trans* del dominio destino, el correspondiente *place* que representa a la molécula en la red de Petri, así como el arco que que relaciona el elemento de tipo *Place* con el elemento de tipo *Trans*. Además, dichos elemento que se crean deben estar contenidos en un elemento de tipo *Page*.

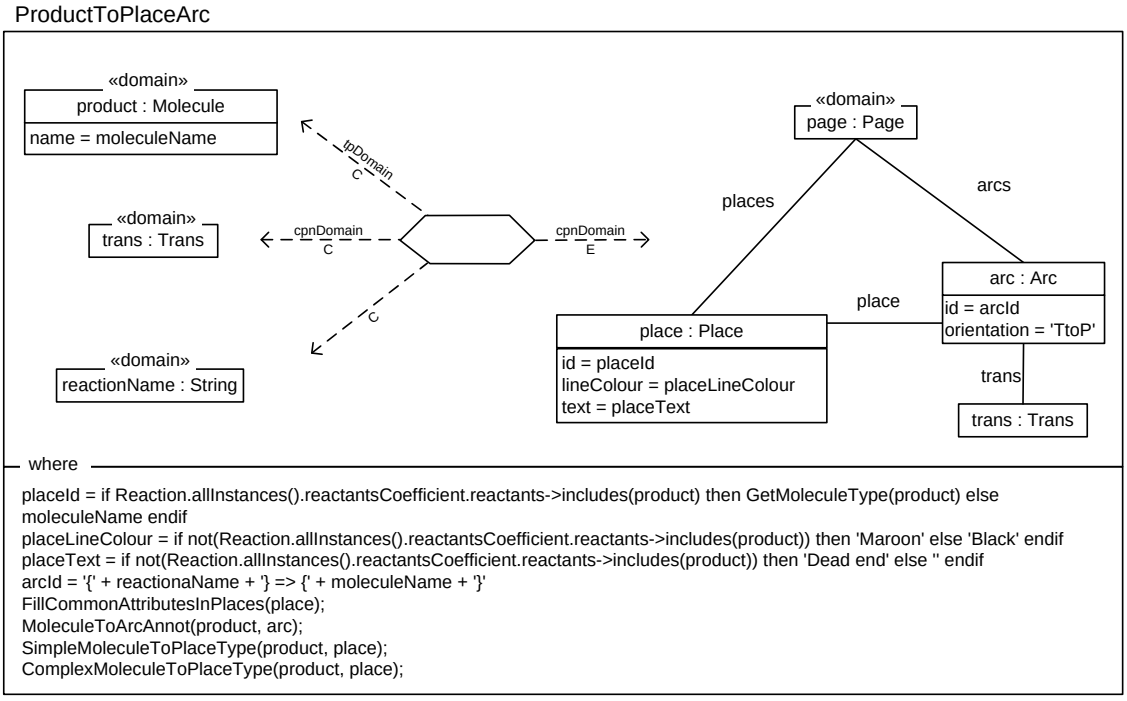


Figura 6.29: Regla *ProductToPlaceArc* en MediniQVT.

6.3.2.7. Regla FillCommonAttributesInPlaces

La regla *FillCommonAttributesInPlaces* 6.30 es una regla *auxiliar* que se invoca desde diferentes reglas para establecer de una forma consistente los atributos de un *Place*.

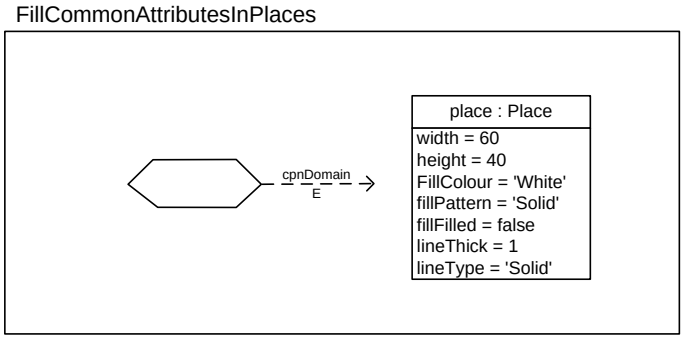


Figura 6.30: Regla *FillCommonAttributesInPlaces* en MediniQVT.

Cuando esta regla con un único dominio se ejecuta, fuerza a que el place que hace *matching* con la variable *place* tenga los atributos que se especifican en la figura 6.30.

6.3.2.8. Regla SimpleReactantToInitMark

La regla *SimpleReactantToInitMark* 6.31 establece para cada *place* generado a partir de una molécula simple la marca inicial del Place (*imark*). Esta marca inicial, establece el número (*atributo* value) de los tokens (*mark*). Igualmente, se establecerá el tipo (color) del token (rol *colorSetElement*).

El valor del atributo *value* se establece de forma automática mediante la expresión OCL `reactant.rkoutsCoefficient.coefficient->sum()`. Este valor es únicamente un cálculo de las moléculas de cierto tipo que son necesarias para que la reacción se lleve a cabo y se pueda simular la red de Petri directamente. No obstante, este valor calculado no tiene validez de cara a las simulaciones, y debe ser establecido finalmente en la red de Petri d forma manual por los expertos basándose en su conocimiento experimental.

6.3.2.9. Regla SimpleMoleculeToPlaceType

La regla mostrada en la figura 6.32 establece el tipo de un place que se corresponde con una molécula de tipo simple del dominio origen. En este caso, el tipo correspondiente para moléculas simples es un elemento de tipo *Enumerated*. Este elemento será único para cada tipo de molécula (obtenido a partir de la expresión OCL *GeTMoleculeType(. . .)* 6.29.

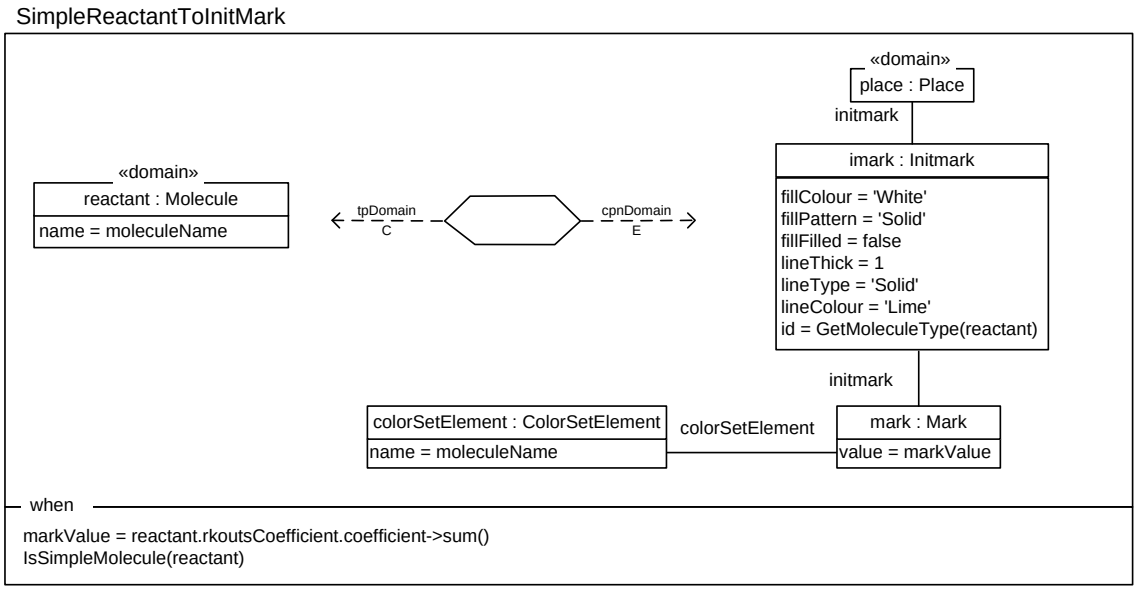


Figura 6.31: Regla *SimpleReactantToInitMark* en MediniQVT.

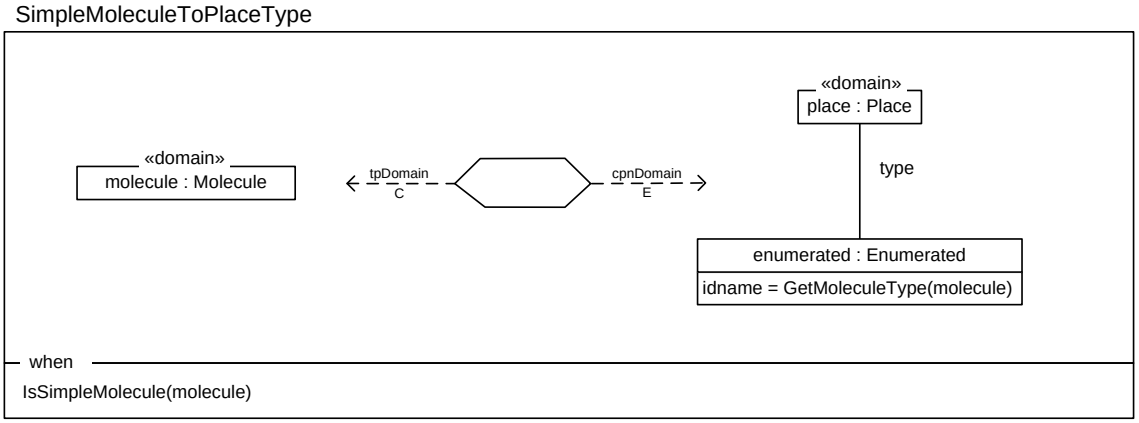


Figura 6.32: Regla *SimpleMoleculeToPlaceType* en MediniQVT.

6.3.2.10. Regla ComplexMoleculeToPlaceType

La regla *ComplexMoleculeToPlaceType* 6.33 es similar a la regla anterior, salvo que en este caso se aplica para moléculas complejas, por lo que el tipo de los places correspondientes a este este tipo de moléculas es un elemento de tipo *Product*.

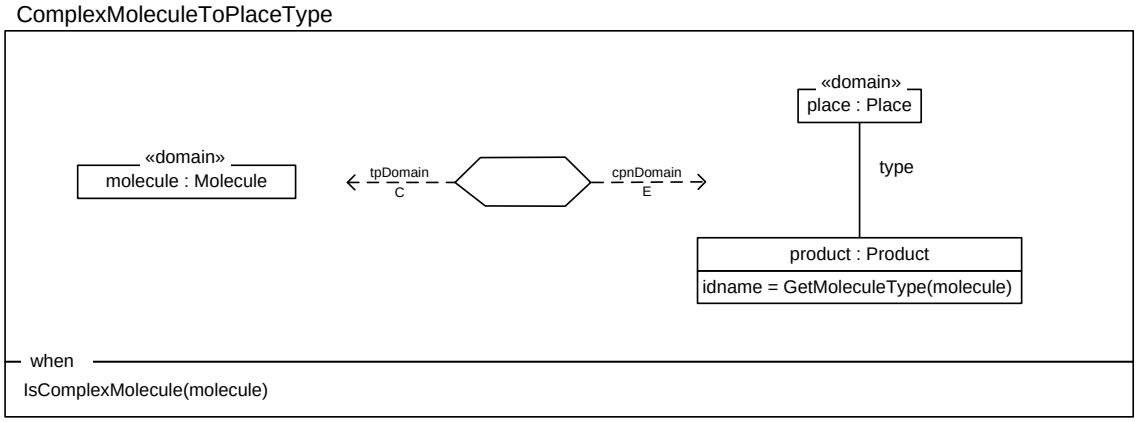


Figura 6.33: Regla *ComplexMoleculeToPlaceType* en MediniQVT.

6.3.2.11. Regla MoleculeToArcAnnot

La regla *MoleculeToArcAnnot* 6.34 se encarga de establecer los atributos de los arcos que unen los elementos de tipo *Place* y los elementos de tipo *Trans*. En este caso, dependiendo de si los places con los que se relacionan provienen de moléculas simples o sencillas, el color de los arcos variará.

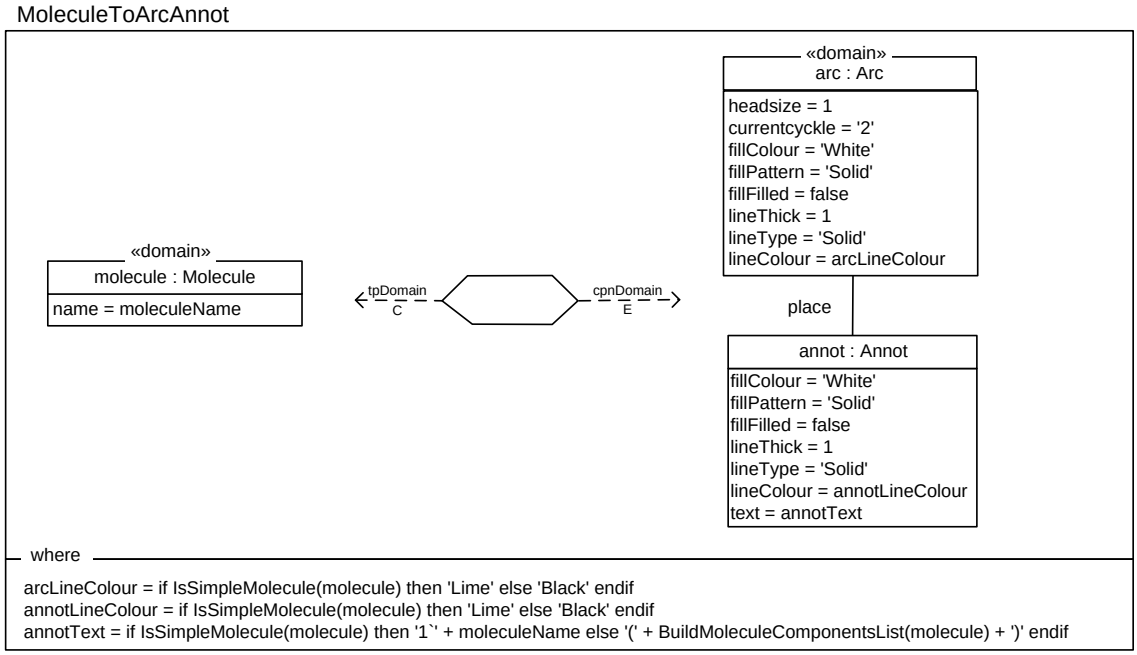


Figura 6.34: Regla *MoleculeToArcAnnot* en MediniQVT.

Además, también se genera la anotación (etiqueta) correspondiente al arco. El color

de la etiqueta variará de la misma forma que el color de los arcos, y el texto de la etiqueta dependerá del tipo de molécula con el que se relaciona el *place*. En caso de que la molécula sea compleja, el texto se construirá con ayuda de la consulta OCL *BuildMoleculeComponentsList(...)* 6.31.

```
1   query BuildMoleculeComponentsList(molecule : Molecule) : String
2   {
3
4     let moleculesSet : OrderedSet(Molecule) =
5       molecule.statesOf
6       ->asSet()
7       ->sortedBy(molecule : Molecule | GetMoleculeType(molecule))
8     in
9       moleculesSet
10      ->excluding(moleculesSet.last())
11      ->iterate(molecule : Molecule; text : String = '' | text.concat(molecule.name).
            concat(','))
12      .concat(moleculesSet.last().name)
13  }
```

Listado 6.31: Query BuildMoleculeComponentsList(...)

6.4. Integración de Medini QVT

6.4.1. Integración del motor de transformaciones Plugin es.upv.dsic.issi.qvt.engine

La herramienta de transformaciones se basa en la biblioteca de transformaciones MediniQVT [33]. Como ya se ha comentado en la sección 4.3, la herramienta MediniQVT es libre, aunque esta licencia sólo se aplica a lo que forma estrictamente el motor de transformaciones, sin serlo ninguna de las herramientas adicionales que proporcionan. Por tanto, para realizar nuestra implementación es necesario definir un plugin que contenga la funcionalidad del motor de transformaciones, y sobre él construir el resto de las herramientas propias. En nuestro caso se ha preferido obtener el código fuente de la biblioteca directamente de los repositorios públicos, y encapsular la funcionalidad completamente en un plugin que llamaremos es.upv.dsic.issi.qvt.engine.

6.4.1.1. Descripción

El plugin es.upv.dsic.issi.qvt.engine contiene todo lo necesario para ejecutar las transformaciones definidas en QVT–Relations. Se ha decidido abordar el diseño partiendo directamente del codigo fuente disponible, en lugar de empleando las versiones precompiladas puesto que se han realizado ligeras modificaciones sobre ellas.

En primer lugar, en las versiones precompiladas, aunque se depende de EMF, en vez

de incluirlo como una dependencia de plugin en el archivo de manifiesto, se ha decidido incluirlo como si fuera una dependecia de una librería común, incluyendo un fichero «*.jar». Para nuestra implementación, en este caso se ha decidido incluir la dependencia empleando los plugins de EMF que estén instalados en la plataforma, evitando posibles problemas debidos a la existencia de diferentes versiones del *framework* de EMF.

Por otra parte, se ha modificado la biblioteca de OCL permitiendo la operación de ordenación de conjuntos de *Strings*, ya que es necesaria emplearla en la transformación.

6.4.1.2. Dependencias

Este plugin tiene dependencias tanto de otros plugins de Eclipse, como de librerías Java comunes. Respecto a los plugins, sus dependencias son:

- org.eclipse.core.runtime
- org.eclipse.emf.common
- org.eclipse.emf.ecore
- org.eclipse.emf.ecore.xmi
- org.eclipse.emf.edit
- org.eclipse.emf.transaction
- org.eclipse.emf.validation

Respecto a las bibliotecas comunes, sus dependencias son:

- Apache Commons Collections [22] (commons-collections-3.2.jar).
- CUP Parser Generator for Java [31] (CUPRuntime.jar).
- Kent Modeling Framework [52] (KMF_Util.jar, KMF_XMI.jar, KMFpatterns.jar, Util-1.2.jar)

6.4.1.3. Listado de paquetes

A continuación se listan todos los paquetes de código fuente que se incluyen en el plugin, indicando aquellos a los que se les han realizado modificaciones.

- de.ikv.emf.qvt
- de.ikv.medini.qvt
- de.ikv.medini.qvt.execution
- de.ikv.medini.qvt.execution.debug

- de.ikv.medini.qvt.execution.debug.events
- de.ikv.medini.qvt.execution.debug.replies
- de.ikv.medini.qvt.execution.debug.requests
- de.ikv.medini.qvt.execution.debug.stackframe
- de.ikv.medini.qvt.model.qvtbase
- de.ikv.medini.qvt.model.qvtbase.impl
- de.ikv.medini.qvt.model.qvtbase.util
- de.ikv.medini.qvt.model.qvtrelation
- de.ikv.medini.qvt.model.qvtrelation.impl
- de.ikv.medini.qvt.model.qvtrelation.util
- de.ikv.medini.qvt.model.qvttemplate
- de.ikv.medini.qvt.model.qvttemplate.impl
- de.ikv.medini.qvt.model.qvttemplate.util
- de.ikv.medini.qvt.qvt
- de.ikv.medini.qvt.qvt.impl
- de.ikv.medini.qvt.qvt.util
- de.ikv.medini.qvt.syntax.parser
- de.ikv.medini.qvt.util
- es.upv.dsic.issi.qvt.engine

  En este paquete se encuentra la clase activadora del plugin, QvtEnginePlugin, a través de la cual se controla el ciclo de vida del éste.
- org.oslo.ocl20
- org.oslo.ocl20.bridge4emf
- org.oslo.ocl20.semantics
- org.oslo.ocl20.semantics.analyser
- org.oslo.ocl20.semantics.bridge
- org.oslo.ocl20.semantics.bridge.impl

- org.oslo.ocl20.semantics.bridge.util
- org.oslo.ocl20.semantics.factories
- org.oslo.ocl20.semantics.impl
- org.oslo.ocl20.semantics.model.contexts
- org.oslo.ocl20.semantics.model.contexts.impl
- org.oslo.ocl20.semantics.model.contexts.util
- org.oslo.ocl20.semantics.model.expressions
- org.oslo.ocl20.semantics.model.expressions.impl
- org.oslo.ocl20.semantics.model.expressions.util
- org.oslo.ocl20.semantics.model.types
- org.oslo.ocl20.semantics.model.types.impl
- org.oslo.ocl20.semantics.model.types.util
- org.oslo.ocl20.semantics.util
- org.oslo.ocl20.standard.lib

  Este paquete contiene la clase OclStringImpl. En esta clase se han añadido los métodos lessThan(...) y greaterThan(...) que se emplean para poder comparar dos *Strings* y ordenarlos sexún su orden lexicográfico.

- org.oslo.ocl20.standard.types

  Este paquete contiene la clase StringTypeImpl. En ella está el método createOperations(...), que ha sido modificado para dar soporte a las operaciones *menor que* y *mayor que*, puesto que son la operación básica sobre la que se sustenta en operador de OCL *sortedBy(...)*.

- org.oslo.ocl20.syntax.ast
- org.oslo.ocl20.syntax.ast.contexts
- org.oslo.ocl20.syntax.ast.contexts.impl
- org.oslo.ocl20.syntax.ast.contexts.util
- org.oslo.ocl20.syntax.ast.expressions
- org.oslo.ocl20.syntax.ast.expressions.impl
- org.oslo.ocl20.syntax.ast.expressions.util

- org.oslo.ocl20.syntax.ast.impl
- org.oslo.ocl20.syntax.ast.qvt
- org.oslo.ocl20.syntax.ast.qvt.impl
- org.oslo.ocl20.syntax.ast.qvt.util
- org.oslo.ocl20.syntax.ast.types
- org.oslo.ocl20.syntax.ast.types.impl
- org.oslo.ocl20.syntax.ast.types.util
- org.oslo.ocl20.syntax.ast.util
- org.oslo.ocl20.syntax.parser
- org.oslo.ocl20.synthesis

6.4.2. Entorno automatizado de ejecución de transformaciones

Para la ejecución de transformaciones se ha desarrollado el plugin es.upv.dsic.issi.qvt.-launcher, que importa el plugin es.upv.dsic.issi.qvt.engine y se encarga de realizar la invocación del método QvtProcessorImpl.evaluateQVT(...).

La funcionalidad de este plugin además, se fundamenta en el código generado de dos modelos Ecore. El primero de ellos se corresponde con el modelo de invocación de transformaciones (sección 6.4.2.1). Este modelo permite especificar a alto nivel los datos de la invocación de una operación. El segundo modelo Ecore (descrito en la sección 6.4.2.2) se corresponde con el metamodelo genérico de trazabilidad. Este metamodelo genérico es propio de nuestra implementación, y permite emplear herramientas genéricas para navegar y consultar las trazas generadas por una transformación en QVT.

6.4.2.1. Especificación de invocaciones

Para poder tratar de forma sencilla con la información de la invocación de una transformación se ha definido un modelo en Ecore para ello. Esto nos permite generar de forma sencilla el código para poder consultar esta información, así como obtener de forma automática los mecanismos de persistencia y recuperación de la información.

De esta manera, empleando el código generado, distintos plugins podrán intercambias fácilmente la información de una invocación. En este caso se empleará dos funcionalidades fundamentalmente:

- En primer lugar, el plugin gráfica que implementa la interfaz de usuario para configurar una nueva transformación emplea el patrón de Modelo/Vista/Controlador.

Empleando EMF el código para la implementación del modelo viene dado de forma gratuita.

- En segundo lugar, la clase que según la API debe lanzar la ejecución de un comando externo (org.eclipse.debug.core.model.ILaunchConfigurationDelegate) debe recibir los datos de la configuración a ejecutar en formato textual. Mediante el mecanismo de persistencia a XMI, la representación de la invocación de forma textual se obtiene de forma automática.

Para el modelo de invocación de transformaciones se han generado los plugins del modelo y del soporte génerico para edición. El plugin del editor no se ha generado puesto que es un modelo principalmente para uso interno y el editor resulta innecesario.

Plugin es.upv.dsic.issi.qvt.launcher.model

Descripción

El plugin es.upv.dsic.issi.qvt.launcher.model contiene el modelo y su código generado para la descripción de invocaciones de transformaciones. En ete caso, el modelo definido se muestra en la figura 6.35.

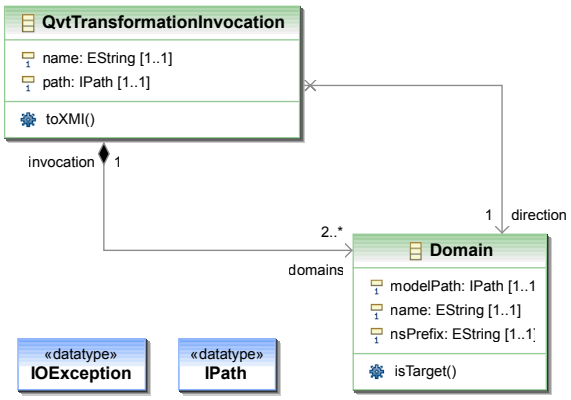


Figura 6.35: Modelo de invocación de transformaciones

En esta figura se observa que el modelo dispone de únicamente dos clases QvtTransformationInvocation y Domain. La primera de ellas se corresponde con la propia transformación, y como se observa, una transformación comprende varios dominios involucrados.

En la primera de las clases además, se almacena toda la información relevante acerca de una transformación (esto es, la ruta del fichero con su descripción textual, y el nombre de la transformación) y en la segunda clase se encuentra la información de cada dominio (en este caso, el nombre que se le asigna al dominio, el *nsPrefix* del metamodelo al que conforma, y la ruta del fichero que contiene la instancia que hará *matching* con este dominio). Además, se observa un rol, *direction*, que enlaza la clase QvtTransformationInvocation

con la clase Domain. Este rol indica en qué dirección se ejecutará la transformación, o lo que es lo mismo, cual es el dominio que se considerará destino de ésta.

Por último, se observan dos tipos de datos propios del modelo Ecore, *IOException* e *IPath*. Estos tipos de datos se corresponden internamente con las clases java.io.IOException y org.eclipse.core.runtime.IPath. La primera se corresponde con la excepción que puede lanzar el método QvtTransformationInvocation.toXMI(...) y no es serializable. La segunda (*IPath*), sí es serializable, y deberá modificarse el código generado de forma automática para que la serialización–deserialización se realice de forma correcta.

Dependencias

El plugin contiene la siguientes dependencias:

- org.eclipse.core.runtime – Por ser un plugin de Eclipse y usar clases e interfaces básicas del entorno como la interfaz *IPath*.
- org.eclipse.core.resources – Por contener las clases que permiten acceder a los recursos del *workspaces* (como la interfaz *IFile*).
- org.eclipse.emf.ecore – Por ser un plugin de Ecore.
- es.upv.dsic.issi.qvt.engine – Este plugin proporciona los mecanismos de análisis de un fichero textual que describe textualmente una transformación en QVT–Relation. Empleándolo es posible crear una nueva instancia de QvtTransformationInvocation (con sus dominios asociados) consultando el resultado de dicho análisis.

Descripción de paquetes y clases

A continuación se listan y se describen los aspectos más relevantes de los paquetes y clases que contiene el plugin.

- Paquete es.upv.dsic.issi.qvt.launcher.model.qvtinvocation

  El paquete es.upv.dsic.issi.qvt.launcher.model.qvtinvocation contiene las interfaces que implementan las clases que se encuentran en el paquete es.upv.dsic.issi.qvt.launcher.model.qvtinvocation.impl. En EMF se implementa tanto una interfaz como una clase por cada una de las clases definidas en el modelo Ecore. De esta manera, se pueden simular en Java los mecanismos de herencia múltiple que permite Ecore. Las interfaces QvtinvocationFactory y QvtinvocationPackage no se generan a partir de las clases del modelo, sino que se corresponden con el modelo en sí. En particular, QvtinvocationFactory es la interfaz que permite acceder a la factoría para crear nuevas instancias de los objetos del modelo, y QvtinvocationPackage se corresponde con la interfaz que contiene la información del paquete (nsPrefix, nsUri, etc.) así como el acceso a la instancia singleton del paquete.

- Domain.java
- QvtinvocationFactory.java
- QvtinvocationPackage.java
- QvtTransformationInvocation.java

- Paquete es.upv.dsic.issi.qvt.launcher.model.qvtinvocation.impl

  El paquete es.upv.dsic.issi.qvt.launcher.model.qvtinvocation.impl contiene las clases que implementan el código del modelo. En este sentido, se crea una clase Java (con el sufijo *–Impl*) para cada una de las clases del modelo. Cada una de esta clases contiene los métodos (con código generado) para consultar, navegar y modificar instancias del modelo (métodos *getters* y *setters*). Estos métodos además, contienen los mecanismos pertinentes para mantener la coherencia entre los objetos del modelo cuando se encuentran relaciones bidireccionales entre las clases del modelo (mediante la interfaz *Notifier*).

  A continuación se muestran las clases que contiene el paquete, así como las modificaciones que se han realizado al código generado.

  - DomainImpl.java

    Como se ha explicado en secciones anteriores, en EMF no se genera el código para los métodos definidos a nivel de modelo. Como se observa en la figura 6.35, en esta clase se ha definido el método isTarget(...), que comprueba si el dominio actual es el dominio destino de la transformación. El listado 6.32 muestra cómo se ha implementado, obsérvese la modificación del *tag* `@generated` como `@generated not` para evitar que se sobreescriba al regenerar el código.

```
1       /**
2       * <!-- begin-user-doc -->
3       * <!-- end-user-doc -->
4       * @generated not
5       */
6       public boolean isTarget() {
7       return (this.equals(getInvocation().getDirection()));
8       }
```

Listado 6.32: Método isTarget(...)

  - QvtinvocationFactoryImpl.java

    Esta es la clase que implementa las factorías de objetos del modelo (en EMF se recomienda usar el patrón de factorías, en lugar de emplear el operador *new* de Java). Estas factorías por defecto tienen un comportamiento muy básico, y fundamentalmente similar al operador new. Por ello, en este caso se ha enriquecido al conjunto de factorías con dos métodos nuevos que se muestran en el listado 6.33.

```
1       public QvtTransformationInvocation createQvtTransformationInvocation(
            String XMI) throws IOException;
2       public QvtTransformationInvocation createQvtTransformationInvocation(IFile
             file);
```

Listado 6.33: Factorías para la clase QvtTransformationInvocation añadidas

La primera de ellas (listado 6.34) permite crear una nueva invocación (incluyendo todos sus hijos, es decir, elementos de tipo *Domain* a partir de una representación textual en XMI. Para ello, se crea un nuevo recurso de EMF en memoria y se carga en él el *stream* que representa a la instancia del modelo empleando los mecanismos de deserialización por defecto de Eclipse. Finalmente, una vez se ha reconstruido en memoria el conjunto de objetos representados por el XMI, se devuelve el elemento raíz, que será la invocación de la transformación.

```
1       public QvtTransformationInvocation createQvtTransformationInvocation(
            String XMI) throws IOException {
2         Resource res = (new ResourceSetImpl()).createResource(URI.createURI("In
              memory resource"));
3         res.load(new ByteArrayInputStream(XMI.getBytes()), null);
4         if (!res.getContents().isEmpty()) {
5           return (QvtTransformationInvocation) res.getContents().get(0);
6         } else {
7           return null;
8         }
9       }
```

Listado 6.34: Método createTransformationInvocation(String XMI)...

El segundo método del listado 6.33 permite crear un elemento de tipo *QvtTransformationInvocation* (incluyendo todos sus hijos) a partir de la representación textual de un programa en *QVT–Relations* textual. Ha de tenerse en cuanta que este fichero únicamente contiene información sobre la transformación, esto es, la ruta del fichero, el nombre de la transformación y el número, nombre y tipo de los dominios involucrados. En este caso, se devolverá la instancia conteniendo los datos conocidos, manteniéndose como indefinidos los datos desconocidos.

En la interfaz de invocación de transformaciones (como se verá en la siguiente sección) se le muestra al usuario una tabla donde cada fila se corresponde con un dominio de la transformación. En ella, el usuario debe de indicar cuáles serán las instancias a transformar (respetando los tipos de los dominios) y la dirección de la transformación. Esta tabla es sólo una representación tabular de una instancia de tipo *QvtTransformationInvocation*.

De esta manera, este método permite inicializar dicha instancia, añadiendo un elemento de tipo *Domain* (e inicializando su *nsPrefix* por cada dominio definido en la representación textual de la transformación. La dirección de la transformación apunta por defecto al último dominio definido en ella. En caso de que no sea así, el usuario puede modificarlo junto con el resto de los datos mediante la interfaz gráfica. El siguiente listado (6.35) muestra la implementación.

```
1   public QvtTransformationInvocation createQvtTransformationInvocation(IFile
        file) {
2
3     QvtTransformationInvocation invocation = null;
4
5     EMFQvtProcessorImpl emfQvtProcessorImpl = new EMFQvtProcessorImpl(new
          ConsoleLog());
6
7     try {
8       TopLevelAS topLevelAS = emfQvtProcessorImpl.parseQvt(new
            InputStreamReader(file.getContents()));
9       EList<TransformationAS> list = topLevelAS.getTransformations();
10      if (list != null) {
11        if (list.size() > 0)  {
12          if (list.size() > 1) {
13            emfQvtProcessorImpl.getLog().printMessage("The file contains
                  more than one transformation!");
14          }
15          if (list.get(0) instanceof TransformationAS) {
16
17            invocation = QvtinvocationFactory.eINSTANCE.
                  createQvtTransformationInvocation();
18            invocation.setName(((TransformationAS)list.get(0)).getName());
19            invocation.setPath(file.getFullPath());
20
21            EList<ModelDeclarationAS> models = ((TransformationAS)list.get
                  (0)).getModelDeclarations();
22
23            Domain domain = null;
24
25            for (ModelDeclarationAS model : models) {
26              domain = QvtinvocationFactory.eINSTANCE.createDomain();
27              domain.setName(model.getModelId());
28              domain.setNsPrefix((String) model.getMetaModelIds().get(0));
29              invocation.getDomains().add(domain);
30            }
31            invocation.setDirection(domain);
32          }
33        }
34      } else {
35        emfQvtProcessorImpl.getLog().printMessage("Invalid QVT
              transformation");
36      }
37    } catch (CoreException e) {
38      e.printStackTrace();
39      emfQvtProcessorImpl.getLog().printMessage("Unable to retrieve file
            contents");
40    }
41    return invocation;
42  }
```

Listado 6.35: Método createTransformationInvocation(IFile file)

Se observa que se crea un elemento de tipo *EMFQvtProcessorImpl*, que es la

clase encargada del análisis de la transformación. Una vez se ha obtenido el resultado del análisis (variable topLevelAS, línea 8), se comprueba que el fichero no estaba vacío (línea 10), que contenía al menos una transformación (línea 11) (sólo se procesará la primera, si contiene más de una se notifica —línea 13—).

A continuación, en las líneas 17–19 se crea un nuevo elemento de tipo *Qvt-TransformationInvocation* a partir de la información que existe en la primera transformación del documento (*list.get(0)*), que es de tipo *TransformationAS*. Tras ello, en la línea 21 se obtiene la lista de dominios de la transformación, y por último, en las líneas 26–29 se crea cada uno de los elementos de tipo *Domain* y se añade a la invocación. El último paso (línea 31) establece la dirección de la transformación hacia el último dominio de ésta.

Por otra parte, además de los métodos de creación de la factoría, esta clase también contiene los métodos para definir los mecanismos de serializacón y deserialización para los tipos de datos desconocidos. En este caso, el tipo *IPath* es desconocido, y deben definirse los métodos convert*Tipo*ToString(...) y create*Tipo*FromString(...).

A continuación, en el listado 6.36 se muestra cómo se puede crear un elemento de tipo *IPath* a partir de su representación textual.

```
1       /**
2        * <!-- begin-user-doc -->
3        * <!-- end-user-doc -->
4        * @generated not
5        */
6       public IPath createIPathFromString(EDataType eDataType, String
            initialValue) {
7         return (IPath)new Path(initialValue);
8       }
```

Listado 6.36: Método convertIPathFromString(...)

El listado 6.37 muestra cómo se representa un elemento de tipo *IPath* como una cadena de texto. En la implementación que se proporciona por defecto se invoca al método clásico Object.toString(), que para un elemento de tipo *IPath* proporciona una representación adecuada ya que realmente es un método que se reescribe en la clase org.eclipse.core.runtime.Path, de forma que no es necesaria modificar esta implementación propuesta.

```
1       /**
2        * <!-- begin-user-doc -->
3        * <!-- end-user-doc -->
4        * @generated
5        */
6       public String convertIPathToString(EDataType eDataType, Object
            instanceValue) {
7         return super.convertToString(eDataType, instanceValue);
```

```
8       }
```

Listado 6.37: Método convertIPathToString(...)

- QvtinvocationPackageImpl.java

  La clase QvtinvocationPackageImpl se corresponde con la clase que implementa el paquete del modelo. De ésta clase sólo habrá una única instancia en memoria (según el patrón *singleton*) y contiene los métodos que permiten recrear en memoria la estructura del modelo al inicializarse el plugin.

- QvtTransformationInvocationImpl.java

  En la clase *QvtTransformationInvocation* se ha definido el método toXMI() que permite proyectar el código XMI que representa a una invocación de una transformación representada como objetos de EMF en memoria. Para ello, lo que se crea es un nuevo *recurso* de EMF en memoria, se le añade el elemento a proyectar (la instancia this), y se invoca al método Resource.save(...) pasándole como argumento un *stream* de bytes que será el valor devuelto (voncertido convenientemente a una cadena de texto). El proceso completo se muestra en el listado 6.38.

```
1       public String toXMI() throws IOException {
2
3         ResourceSet resourceSet = new ResourceSetImpl();
4
5         Resource res = resourceSet.createResource(URI.createFileURI("In memory
              resource"));
6
7         res.getContents().add(this);
8
9         OutputStream os = new ByteArrayOutputStream();
10
11        res.save(os, Collections.EMPTY_MAP);
12
13        return os.toString();
14      }
```

Listado 6.38: Método QvtTransformationInvocation.toXMI()

- Paquete es.upv.dsic.issi.qvt.launcher.model.qvtinvocation.util

  En este paquete se encuentran únicamente dos clases de utilidades que realizan tareas auxiliares comunes en los plugins de EMF para la creación y navegación de los elementos del modelo.

  - QvtinvocationAdapterFactory.java
  - QvtinvocationSwitch.java

- Paquete es.upv.dsic.issi.qvt.launcher.model.qvtinvocation.validation

Este paquete realmente no realiza ninguna acción y puede ignorarse. Simplemente declara una serie de artefactos que declaran interfaces de validación. Este código se crea como ejemplo de cómo se pueden extender las capacidades de generación de código de EMF.

- DomainValidator.java
- IOExceptionValidator.java
- QvtTransformationInvocationValidator.java

Plugin es.upv.dsic.issi.qvt.launcher.model.edit

El plugin es.upv.dsic.issi.qvt.launcher.model.edit proporciona la funcionalidad básica para representar los elementos del modelo de invocaciones en un editor. Puesto que su implementación es la estándar, similar a las explicadas anteriormente, y las modificaciones no son significativas, no se describirán en mayor detalle.

6.4.2.2. Tratamiento de la trazabilidad

En MediniQVT el tratamiento de la trazabilidad se hace según las recomendaciones del estándar de QVT, esto es, se genera una «clase de traza» por cada regla de la transformación. Esta clase de traza tendrá una propiedad por cada una una de las variables libres con las que se pueda hacer *matching*. De esta manera, en la ejecución se genera un objeto de cada una de las clases de trazas por cada aplicación de la regla, quedando sus atributos establecidos a los objetos que hacen matching con los dominios y variables de la regla.

No obstante, en MediniQVT no se proporciona ningún tipo de herramienta para inspeccionar los modelos de trazabilidad. Además, la variabilidad introducida por la existencia de un metamodelo de trazabilidad dependiente de los metamodelos de los dominios involucrados, dificulta la creación de un editor genérico para la visualización y navegación de estos modelos. Es por ello, que se ha decidido adaptar el metamodelo de trazabilidad empleado en MOMENT [5], así como sus herramientas y editores (ampliamente explicados en [26]), para permitir tratar con los modelos de trazabilidad generados por MediniQVT.

A lo largo de las siguientes subsecciones se describen los tres plugins desarrollados para dar soporte al metamodelo de trazabilidad. Ha de tenerse en cuenta que el código se ha rehecho partiendo de una nueva definición del metamodelo, obteniendo un código más claro, elegante, limpio y haciendo mejor uso de las APIs básicas de Eclipse. No obstante, el metamodelo que aquí se presenta es muy similar al descrito detalladamente en [26], por lo que no se entrará a describir en detalle ninguna de las modificaciones realizadas. En su conjunto, los cambios no aportan ninguna novedad a los ya descrito en tal documento e incluso en algunos casos, se ha preferido recortar funcionalidad supérflua (como los asistentes de creación de un nuevo modelo de trazabilidad, o la personalización y extensión de metamodelos).

Plugin es.upv.dsic.issi.traceability.metamodel

Descripción

El plugin es.upv.dsic.issi.traceability.metamodel es el que contiene el código generado para el metamodelo de trazabilidad mostrado en la figura 6.36.

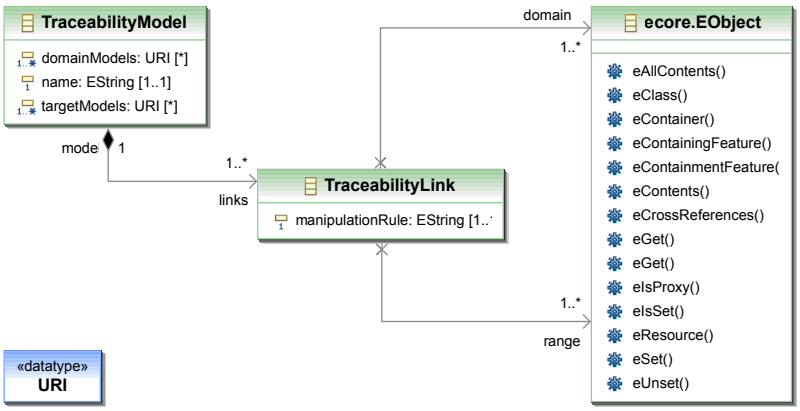


Figura 6.36: Metamodelo de trazabilidad desarrollado para MediniQVT

Como se observa, la clase *raíz* del modelo es la clase *TraceabilityModel*. Un metamodelo de trazabilidad, por tanto, tendrá un nombre y un conjunto de URIs (*domainModels* y *targetModels*) de los dominios origen y destino. Aunque esto no es estrictamente necesario, es útil para poder representar directamente el modelo de trazabilidad en el editor en árbol con tres paneles (que se explicará en la sección 6.4.2.2) ya que no es necesario recorrer los *mappings* del modelo para poder mostrar los elementos de los modelos origen y destino. Para referenciar los modelos origen y destino se emplean elementos de tipo URI (que se corresponden con la clase org.eclipse.emf.common.util.URI puesto que éstos son *recursos* de EMF, y la URI es el mecanismo estándar para cargarlos en memoria y referenciarlos. Es destacable también que un modelo de trazabilidad puede relacionar n modelos origen con n modelos destino.

Además se observa que un modelo de trazabilidad se compone por un conjunto de *mappings* (*TraceabilityLinks*). Un *mapping* o «enlace de trazabilidad» tiene un nombre (que viene dado por la regla que lo genera), un conjunto de elementos dominio (que serán instancias de las clases de los modelos origen) y un conjunto de elementos rango (que a su vez serán instancias de las clases del modelo destino). Para mantener la genericidad, los elementos origen y destino de un determinado *mapping* serán elementos de tipo *EObject* (es decir, cualquier instancia de un modelo de EMF).

Dependencias

Las dependencias en este caso son las comunes de un plugin de código generado de EMF.

- org.eclipse.core.runtime
- org.eclipse.emf
- org.eclipse.emf.ecore

Descripción de paquetes y clases

- Paquete traces

  El paquete traces contiene las interfaces que implementan las clases que se encuentran en el paquete traces.impl. Como ya se ha comentado, en EMF se implementa tanto una interfaz como una clase por cada una de las clases definidas en el modelo Ecore. Recuérdese que también se crea una interfaz para el paquete, y otra para acceder a las factorías. El paquete consta de las siguiente clases:

  - TraceabilityLink.java
  - TraceabilityModel.java
  - TracesFactory.java
  - TracesPackage.java

- Paquete traces.impl

  El paquete traces.impl contiene las clases que implementan el código ejecutable. En este sentido, se crea una clase Java (con el sufijo *–Impl*) para cada una de las clases del modelo. Cada una de esta clases contiene los métodos (con código generado) para consultar, navegar y modificar instancias del modelo (métodos *getters* y *setters*). Estos métodos además, contienen los mecanismos pertinentes para mantener la coherencia entre los objetos del modelo cuando se encuentran relaciones bidireccionales entre las clases del modelo (mediante la interfaz *Notifier*).

  Nótese que en este caso, como se ha empleado un tipo propio al modelo (*URI*), en los métodos de la factoría (*TracesFactoryImpl*) se deberá establecer el mecanismo de serialización y deserialización para las

- instancias de este tipo (métodos createURIFromString(...) y convertURIToString(...)) tal y como se realizó en la sección 6.4.2.1.

  - TraceabilityLinkImpl.java
  - TraceabilityModelImpl.java
  - TracesFactoryImpl.java
  - TracesPackageImpl.java

- Paquete traces.util

  En este paquete se encuentran únicamente dos clases de utilidades que realizan tareas auxiliares comunes en los plugins de EMF para la creación y navegación de los elementos del modelo.

- TracesAdapterFactory.java
- TracesSwitch.java

- Paquete traces.validation

  Este paquete realmente no realiza ninguna acción y puede ignorarse. Simplemente declara una serie de artefactos que declaran interfaces de validación. Este código se crea como ejemplo de cómo se pueden extender las capacidades de generación de código de EMF.

  - TraceabilityLinkValidator.java
  - TraceabilityModelValidator.java

Plugin es.upv.dsic.issi.traceability.metamodel.edit

El plugin es.upv.dsic.issi.traceability.metamodel.edit proporciona la funcionalidad básica para representar los elementos del modelo de invocaciones en un editor. Puesto que su implementación es la estándar, similar a las explicadas anteriormente, y las modificaciones no son significativas, no se describirán en mayor detalle.

Plugin es.upv.dsic.issi.traceability.metamodel.editor

Este es el último plug-in desarrollado para la gestión de la trazabilidad. Engloba el editor de trazabilidad así como el asistente necesario para crear un nuevo modelo de trazabilidad de forma automática. Este plugin proporciona un editor que permite visualizar los modelos de trazabilidad junto con sus modelos dominio y rango de forma simultánea. Además, permite navegar estos modelos de trazabilidad, mostrando de forma automática y de forma sencilla las relaciones entre los elementos de estos modelos dominio y rango. Estos modelos de trazabilidad también pueden ser editados mediante este mismo modelo. El único requisito que tiene el editor para poder representar un modelo de trazabilidad que relaciona cualquier modelo origen con cualquier modelo destino es que sus respectivos metamodelos se encuentren registrados previamente en EMF.

Por último, y aunque para la herramienta basada en MediniQVT no es una característica especialmente útil, proporciona los mecanismos para la creación de forma manual de un modelo de trazas.

Dependencias

El plugin tiene directamente las siguientes dependencias:

- org.eclipse.core.runtime
- org.eclipse.core.resources

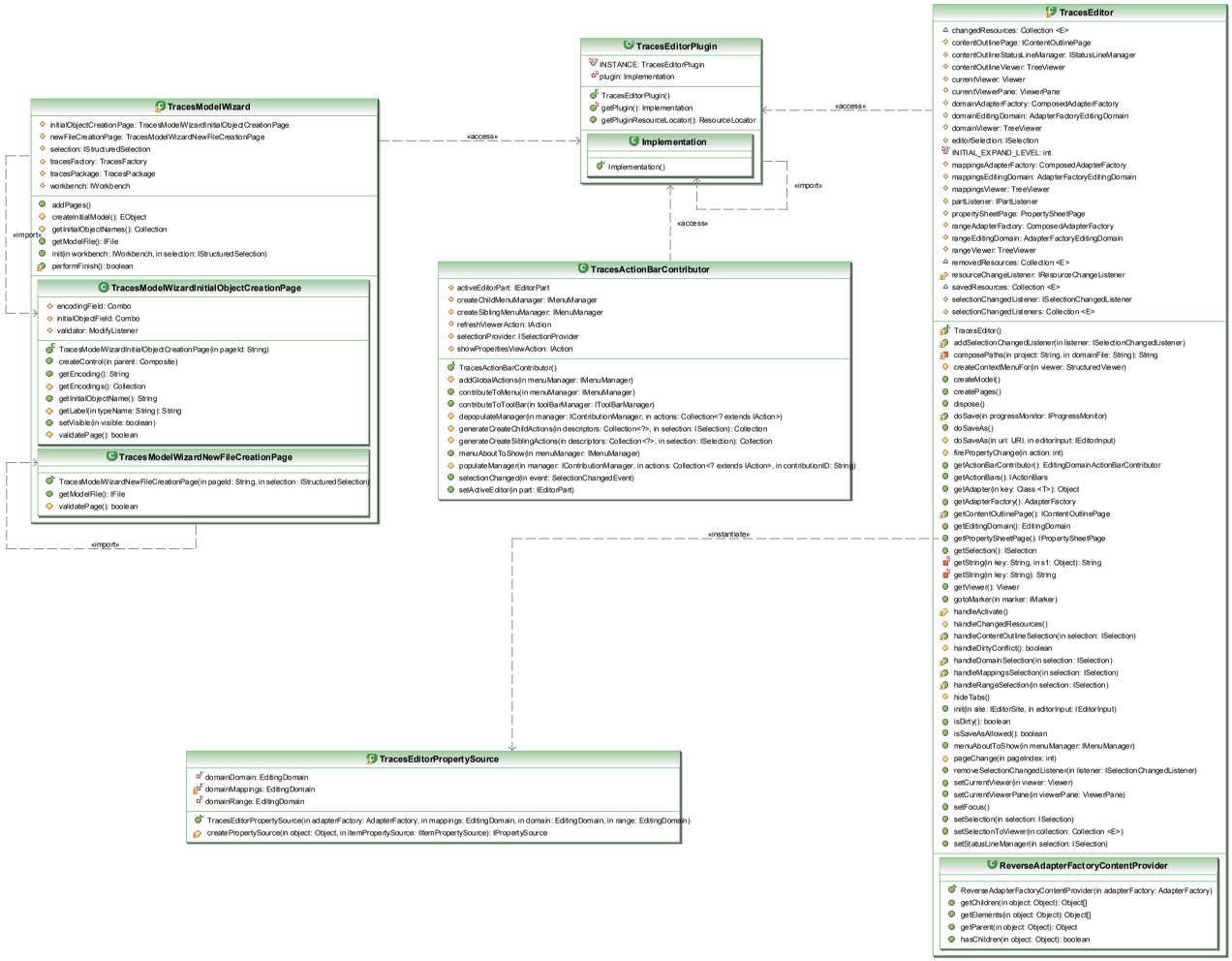


Figura 6.37: Diagrama de clases del paquete traces.presentation.

- es.upv.dsic.issi.traceability.metamodel.edit

  Este plugin a su vez reexporta a es.upv.dsic.issi.traceability.metamodel, que es requerido igualmente.

- org.eclipse.emf.ecore.xmi
- org.eclipse.emf.edit.ui
- org.eclipse.ui.ide

A su vez, y tal y como se explicaba antes, para su correcto funcionamiento dependerá en ejecución de que los paquetes de los metamodelos origen y destino estén correctamente registrados en EMF.

Descripción de paquetes y clases

El plugin contiene un único paquete traces.presentation. La figura 6.37 muestra las clases que éste contiene, así como sus interrelaciones. A continuación se describen sus detalles más significativos.

- TracesEditorPlugin.java

  Esta es la clase activadora del plugin, y como es habitual, se encarga de controlar el ciclo de vida de éste. En este caso, contiene la implementación estándar.

- TracesActionBarContributor.java

  Esta clase implementa la contribución del editor a la barra de menús y de herramientas cuando éste está abierto. Esta contribución se indica en el atributo *contributorClass* en la conexión al punto de extensión *org.eclipse.ui.editors*.

```
1       <extension point = "org.eclipse.ui.editors">
2         <editor
3           id = "traces.presentation.TracesEditorID"
4           name = "%_UI_TracesEditor_label"
5           icon = "icons/full/obj16/TracesModelFile.gif"
6           extensions = "traces"
7           class = "traces.presentation.TracesEditor"
8           contributorClass="traces.presentation.TracesActionBarContributor" >
9         </editor>
10      </extension>
```

Listado 6.39: Conexión al punto de extensión del editor de trazabilidad

La figura 6.38 muestra el aspecto final del menú contribuído.

- TracesEditor.java

  Esta clase implementa el editor de trazabilidad. Se ha partido del código inicial generado de forma automática para obtener su implementación. En este sentido, se

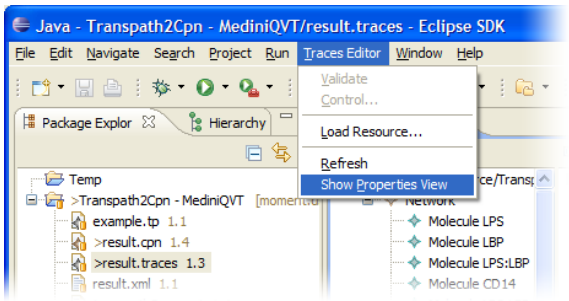


Figura 6.38: Contribución a la barra de menús del editor de trazabilidad

han eliminado las diferentes pestañas que eran innecesarias (por defecto es un editor multipágina), así como la configuración visual, ya que debe mostrar simultáneamente tres árboles de modelos en pantalla.

También se ha modificado el comportamiento del editor, modificando y añadiendo

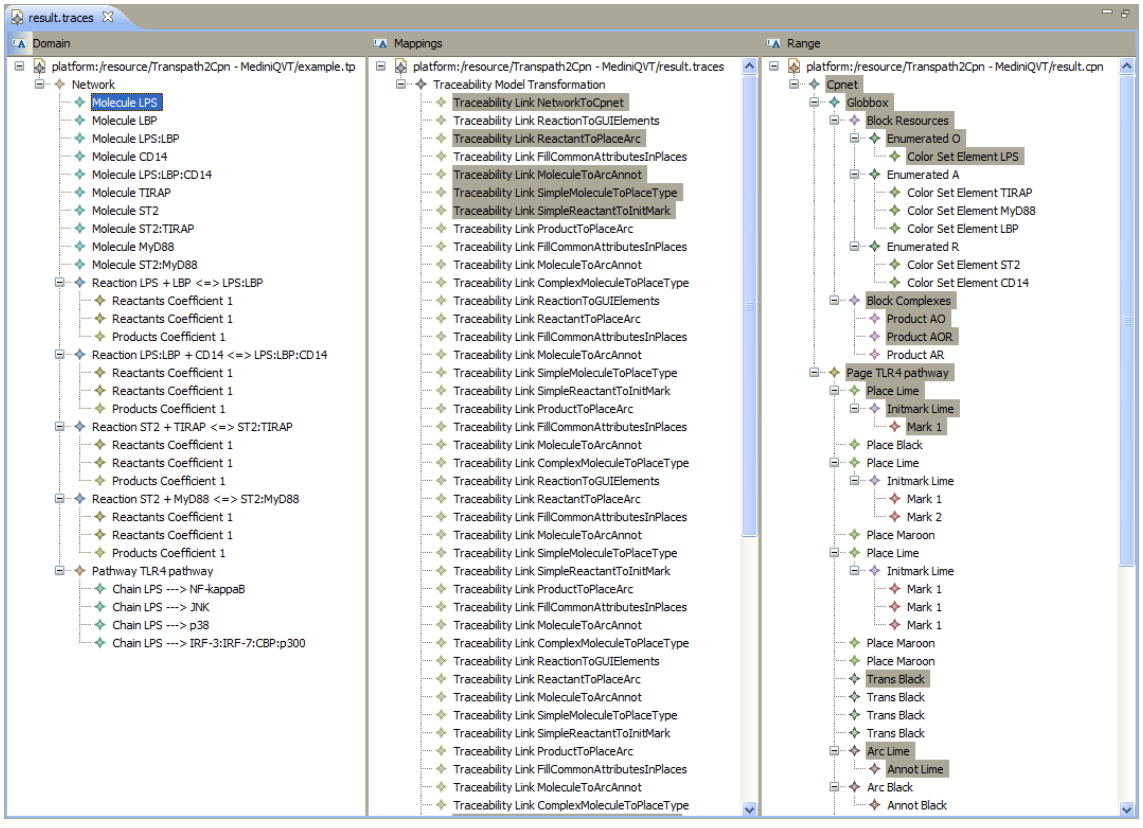


Figura 6.39: Navegación en el editor de trazabilidad a partir de un elemento del modelo origen

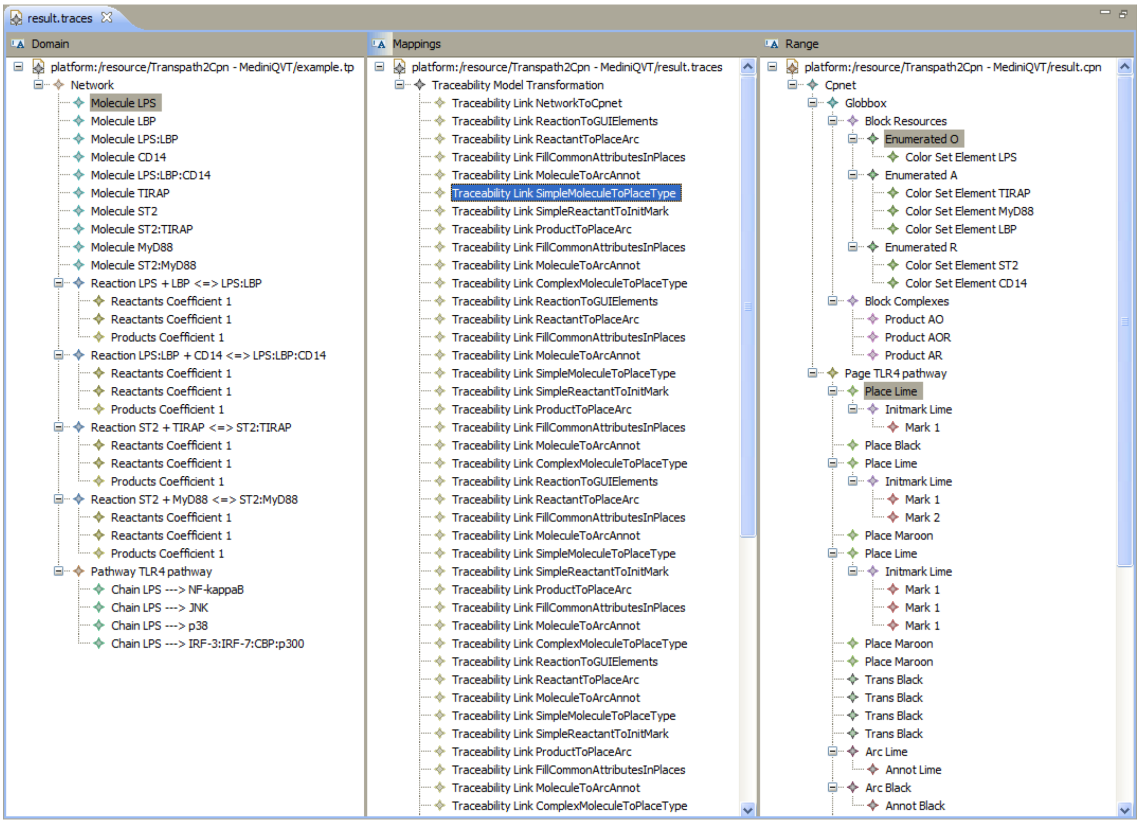


Figura 6.40: Navegación en el editor de trazabilidad a partir de un *mapping*

los métodos para controlar su comportamiento en los cambios de selección. Cuando se selecciona uno o varios elementos de cualquiera de los tres modelos, se navegan los dos restantes, y se resaltan los elementos de éstos relacionados con el/los elemenos seleccionados.

Como ejemplo, la figura 6.39 muestra cómo el elemento «Molecule LPS» ha participado en las reglas *NetworkToCpnet*, *ReactantToPlaceArc*, *MoleculeToArcAnnot*, *SimpleMoleculeToPlaceType* y *SimpleReactantToInitMark*. Fruto de la aplicación de esas reglas, se observa que se han creado todos los elementos resaltados en gris del panel derecho.

En el caso de la figura 6.40 se observa que en la aplicación de la regla *SimpleMoleculeToPlaceType*, se ha creado un *Place* color lima de tipo *Enumerated O* a partir de la molécula *LPS*.

Por último, en la figura 6.41, comprobamos que si queremos saber a partir de qué elemento se crea el *ColorSetElement LPS*, basta con seleccionarlo. De esta manera, se observa que éste se ha creado en la regla *SimpleReactantToInitMark* a partir de la *molécula LPS*.

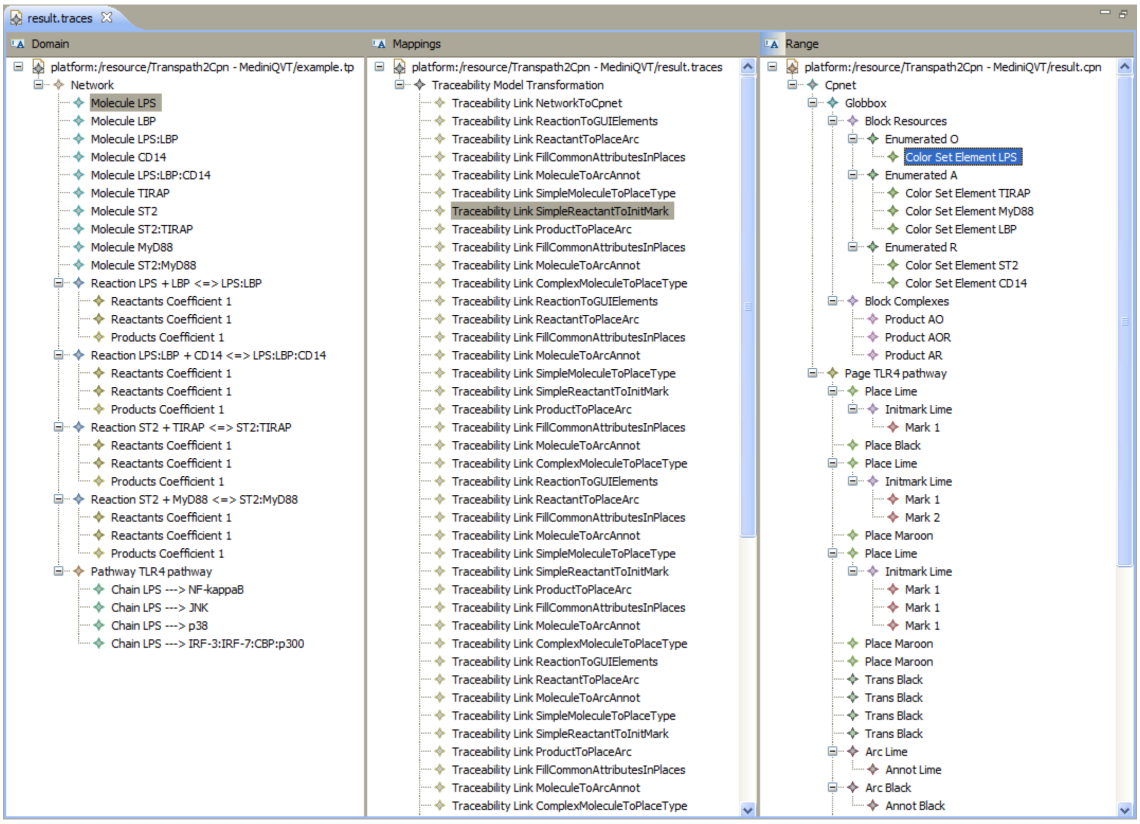


Figura 6.41: Navegación en el editor de trazabilidad a partir de un elemento del modelo destino

- TracesEditorPropertySource.java

  La clase TracesEditorPropertySource, que hereda de org.eclipse.emf.edit.ui.provider.-AdapterFactoryContentProvider, es la clase encargada de gestionar los datos mostrados en la hoja de propiedades del editor. Ésta extiende la clase empleada por defecto de forma que modifica su comportamiento. Esto se debe a que cuando se selecciona un elemento TraceabilityLink, en la hoja de propiedades se pueden establecer qué elementos son los elementos dominio y cuales son los elementos rango. Esto se hace mediante la selección de éstos en una ventana de diálogo. Por defecto, ésta ventana muestra todos los elementos de los modelos cargados en memoria, por lo que podrían añadirse a la lista de elementos dominio elementos del modelo rango, creando un modelo incoherente. Extendiendo la clase org.eclipse.emf.edit.ui.provider.AdapterFactory.ContentProvider podemos realizar un filtrado de los elementos que se van a mostrar, permitiendo que solo se seleccionen aquellos válidos. En la figura 6.37 se observa que es en el método TracesEditor.getPropertySheetPage(. . .) donde se instancia esta clase personalizada.

  En la figura 6.42 se observa que al editar los elementos dominio del *mapping* «*SimpleMoleculeToPlaceType*» en la ventana de selección sólo se encuentran disponibles

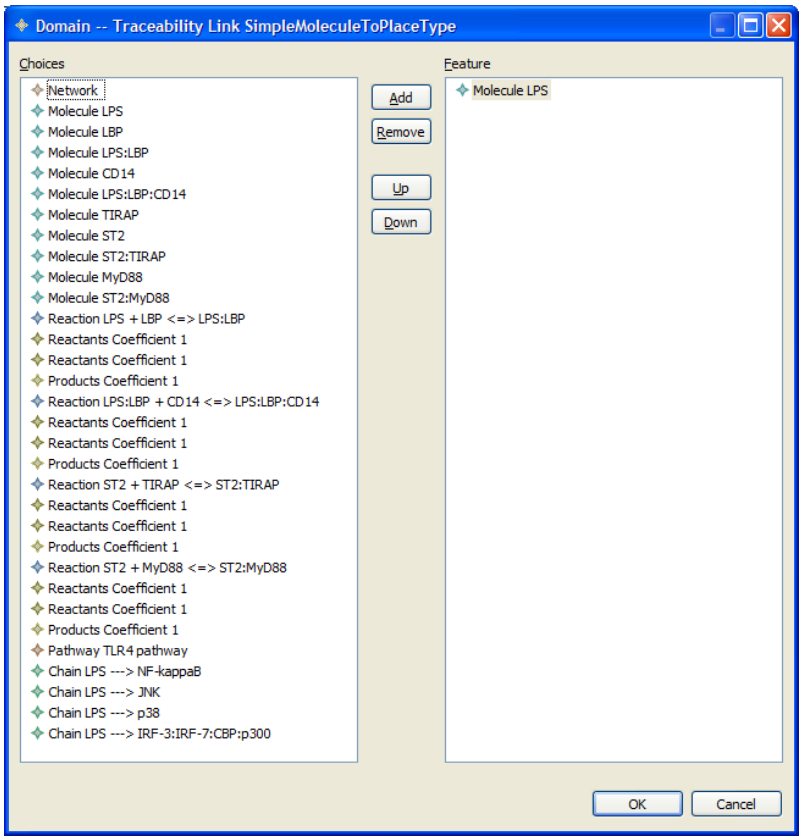


Figura 6.42: Ventana de modificación de los elementos dominio de un *TraceabilityLink*

los elementos del modelo origen.

- TracesModelWizard.java

  La clase *TracesModelWizard* implementa el asistente de creación de un nuevo modelo de trazabilidad. Esta contribución se indica en el atributo *class* en la conexión al punto de extensión *org.eclipse.ui.newWizards* y está implementada de la forma habitual, por lo que no se describirá en mayor detalle.

```
1       <extension
2         point = "org.eclipse.ui.newWizards">
3         <category
4           id = "org.eclipse.emf.ecore.Wizard.category.ID"
5           name="%_UI_Wizard_category">
6         </category>
7         <wizard
8           id = "traces.presentation.TracesModelWizardID"
9           name = "%_UI_TracesModelWizard_label"
10          class = "traces.presentation.TracesModelWizard"
```

```
11          category = "org.eclipse.emf.ecore.Wizard.category.ID"
12          icon = "icons/full/obj16/TracesModelFile.gif">
13          <description>%_UI_TracesModelWizard_description</description>
14          <selection class = "org.eclipse.core.resources.IResource" />
15          </wizard>
16      </extension>
```

Listado 6.40: Conexión al punto de extensión del asistente para modelos trazabilidad

6.4.2.3. Creación y ejecución de trabajos de transformaciones Plugin es.upv.dsic.issi.qvt.launcher

En las secciones anteriores se han mostrado y descrito los plugins que son prerrequisito para poder ejecutar las transformaciones de forma automatizada. En esta sección se mostrará cómo se hace uso de ellos para hacer la llamada a la biblioteca de MediniQVT y obtener los resultados.

Descripción

El plugin es.upv.dsic.issi.qvt.launcher permite, a partir de la especificación de una transformación empleando el modelo definido en es.upv.dsic.issi.qvt.launcher.model, ejecutar la transformación con los argumentos definidos sin intervención del usuario.

La lógica de este plugin se realizará empleando la API para ejecución de aplicaciones externas de Eclipse. En esta API, se define una clase que implementa la interfaz org.eclipse.debug.core.model.ILaunchConfigurationDelegate. Esta clase será la encargada de ejecutar una determinada configuración de ejecución en su método launch(...) (que puede especificarse de diversos modos, en este caso lo será mediante el plugin de interfaz que se describirá en el siguiente apartado «Interfaz gráfica para la ejecución de transformaciones Plugin es.upv.dsic.issi.qvt.launcher.ui») cuando se utilice la interfaz de «*Run configurations...*» de Eclipse. En el listado 6.41 se muestra la conexión al punto de extensión org.eclipse.debug.core.launchConfigurationTypes para que Eclipse sea capaz de encontrar la clase que debe lanzar la ejecución.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <?eclipse version="3.2"?>
3   <plugin>
4     <extension
5       point="org.eclipse.debug.core.launchConfigurationTypes">
6       <launchConfigurationType
7         delegate="es.upv.dsic.issi.qvt.launcher.internal.QvtLaunchConfiguration"
8         id="es.upv.dsic.issi.qvt.launcher.launchConfigurationType"
9         modes="run"
10        name="QVT Transformation">
11      </launchConfigurationType>
12    </extension>
13    <extension
```

```
14          point="org.eclipse.debug.ui.launchConfigurationTypeImages">
15          <launchConfigurationTypeImage
16            configTypeID="es.upv.dsic.issi.qvt.launcher.launchConfigurationType"
17            icon="icons/transf_invocation.png"
18            id="es.upv.dsic.issi.qvt.launcher.launchConfigurationTypeImage">
19          </launchConfigurationTypeImage>
20        </extension>
21      </plugin>
```

Listado 6.41: Fichero es.upv.dsic.issi.qvt.launcher/plugin.xml

Dependencias

El plugin tiene las siguientes dependencias:

- org.eclipse.core.runtime

  Depende de este plugin para poder ejecutarse en la plataforma Eclipse y tener acceso a las APIs básicas

- org.eclipse.debug.core

  En este plugin se definen las clases e interfaces que dan soporte a la ejecución de configuraciones, como la interfaz *ILaunchConfigurationDelegate*.

- org.eclipse.emf.ecore.xmi

  Este plugin se requiere para ser capaz de cargar recursos EMF persistidos en disco.

- es.upv.dsic.issi.qvt.engine

  Este es el plugin que encapsula el motor de transformaciones.

- es.upv.dsic.issi.qvt.launcher.model

  Mediante este plugin seremos capaces de recorrer y consultar la descripción de la invocación de una transformación de forma sencilla empleando el código generado por EMF.

- es.upv.dsic.issi.traceability.metamodel

  Esta última dependencia permite crear y manipular modelos de trazabilidad del metamodelo de trazabilidad específico que se ha definido en la sección anterior a partir de la información devuelta por el motor de transformaciones de MediniQVT.

Descripción de paquetes y clases

Paquete es.upv.dsic.issi.qvt.launcher

Este paquete únicamente contiene la clase activadora del plugin, QvtLauncherPlugin. Esta clase contiene una serie de constantes (cadenas de texto) que sirven como claves para consultar una tabla *hash*. Esto es así porque un elemento de tipo ILaunchConfiguration

(que es uno de los argumentos que se recibe en el método ILaunchConfigurationDelegate.launch(...)) es básicamente eso, una tabla *hash* que contiene la información necesaria para lanzar la ejecución.

Salvo estas constantes, la clase QvtLauncherPlugin contiene la implementación estándar de una clase activadora.

Paquete es.upv.dsic.issi.qvt.launcher.internal

- QvtLaunchConfiguration.java

  La clase QvtLaunchConfiguration es la encargada de recuperar la configuración de una invocación de un proceso externo, y lanzar el proceso de ejecución. Todo esto se hace implementando el método *launch(...)* definido en la interfaz ILaunchConfigurationDelegate.

```
1   public void launch(ILaunchConfiguration configuration, String mode,
2     ILaunch launch, IProgressMonitor monitor) throws CoreException {
3
4     String XMI = configuration.getAttribute(QvtLauncherPlugin.INVOCATION_XMI,"");
5
6     Resource res = (new ResourceSetImpl()).createResource(URI.createURI("In
          memory resource"));
7
8     try {
9       res.load(new ByteArrayInputStream(XMI.getBytes()), null);
10    } catch (IOException e) {
11      e.printStackTrace();
12    }
13
14    if (!res.getContents().isEmpty()) {
15      QvtTransformationInvocation invocation = (QvtTransformationInvocation) res.
            getContents().get(0);
16
17      QvtTransformationProcess transformationProcess = new
            QvtTransformationProcess("",launch);
18
19      launch.addProcess(transformationProcess);
20
21      transformationProcess.start(invocation);
22    }
23  }
```

Listado 6.42: Método QvtLaunchConfiguration.launch(...)

El listado 6.42 muestra cómo se ha implementado este método. En primer lugar (línea 4) se obtiene la configuración de la invocación (en este caso, se opta por guardar toda la información compantada en una cadena de texto que representa el modelo en XMI). Tras ello, se carga en una recurso de EMF en memoria (líneas 6–12). En caso de que el XMI represente un modelo no vacío (línea 14), se accede a la instancia

raíz del modelo (la invocación de la transformación QVT, línea 15), y se crea un nuevo proceso para esa invocación (línea 17). La clase QvtTransformationProcess, que se comenta a continuación, implementa la interfaz org.eclipse.debug.core.model.IProcess. Esto es así porque el elemento de tipo ILaunch, que es el que controlará la ejecución, espera un objeto de este tipo (línea 19). Finalmente, se invoca al método QvtTransformationProcess.start(...) que comienza la ejecución (línea 21).

- QvtTransformationProcess.java

  Esta clase hereda de IProcess, que es el tipo que Eclipse espera que controle el proceso de ejecución. No obstante, no es ella la que ejecuta la invocación directamente, sino que delega este trabajo a un elemento de tipo org.eclipse.core.resources.WorkspaceJob. En el listado 6.43 se muestra cómo se implementa el método start(...).

```
1       public synchronized void start(QvtTransformationInvocation invocation) {
2         thread = new QvtTransformationJob(invocation);
3         IJobManager jobMan = Job.getJobManager();
4         try {
5           jobMan.join("QVT", null);
6         } catch (OperationCanceledException e) {
7           e.printStackTrace();
8         } catch (InterruptedException e) {
9           e.printStackTrace();
10        }
11        thread.schedule();
12      }
```

Listado 6.43: Método QvtTransformationProcess.start(...)

La clase QvtTransformationJob, que se comenta a continuación, es la que hereda de WorkspaceJob y ejecuta finalmente el proceso de transformación.

- QvtTransformationJob.java Esta clase hereda de org.eclipse.core.resources.WorkspaceJob, y encapsulará un trabajo que modificará los recursos del *workspace* (en este caso, creará/modificará ficheros de resultado). Se emplea esta clase puesto que evita que se envíe numerosas veces a los distintos *listeners* de los recursos del *workspace* el evento de que el recurso ha cambiado.

  La clase tiene un atributo invocation, de tipo QvtTransformationInvocation, que contiene los datos para podr llamar a la biblioteca de MediniQVT. Los métodos que implementa son:

  - runInWorkspace(IProgressMonitor)

    Este método es el que consulta el atributo invocation y realiza las siguientes acciones:

    1. Crea una instancia de de.ikv.emf.qvt.EMFQvtProcessorImpl.EMFQvtProcessor (emfQvtProcessorImpl) de la biblioteca de MediniQVT. Esta instancia es la que contendrá todos los datos de la transformación, y finalmente la ejecutará.

2. Después recorre cada uno de los dominios definidos en la invocación de la transformación. Si el fichero que se corresponde con el dominio existe, lo carga como recurso de EMF en memoria y añade su metamodelo a la variable emfQvtProcessorImpl empleando el método addMetamodel(EPackage).
   Si el fichero no existe, o es un recurso EMF vacío, creará un nuevo recurso. En este caso se consultará el atributo *nsPrefix* para determinar su metamodelo consultando el registro de EMF y será añadido igualmente a la lista de metamodelos empleando addMetamodel(EPackage).
3. Si todo ha ido correcto, en cualquiera de las dos posibilidades anteriores, el recurso EMF asociado a dicho dominio se añadirá a una lista ordenada de modelos definida como Collection<Resource>models = new ArrayList<Resource>();. Esta lista será pasada a emfQvtProcessorImpl en el momento de la invocación.
4. A continuación, se accede al recurso del modelo destino, y se elimina el contenido (en este caso, no soportamos transformaciones *in place*, ya que pueden resultar problemáticas en caso de que el modelo preexistente sea erróneo).
5. Finalmente, una vez se han preparado todos los recursos, se le indica a la variable emfQvtProcessorImpl cuáles son los modelos, y se invoca el método evaluateQvt (listado 6.44);
   Obsérvese que como resultado de la invocación del método se devuelve un conjunto de trazas (en la variable traces).

```
1       Collection<Trace> traces = emfQvtProcessorImpl.evaluateQVT(
2         qvtScriptReader,
3             // Representación textual de la transformación
4         invocation.getName(),
5             // Nombre de la transformación a ejecutar
6         true,
7             // Ejecutar en modo enforce (sino, sólo ejecuta check)
8         invocation.getDirection().getName(),
9             // Nombre del dominio destino
10        models.toArray(),
11            // Lista de los modelos que han de corresponderse con los
                  dominios
12        new ArrayList<Trace>(),
13            // Conjunto de trazas previas, si el modelo destino no es vacío
                   y proviene de una ejecución previa
14        log);
15            // Log para guardar mensajes
```

Listado 6.44: Método evaluateQvt(...)

6. Empleando el conjunto de trazas devuelto, se llama a la función QvtTransformationJob.createTraceabilityModel(...) que se explica a continuación.
7. Por último, se salvan el modelo resultado y el modelo de trazabilidad recién creado. El modelo de trazabilidad se guardará en la misma ruta que el

fichero resultado. El nombre del fichero será el mismo que el del modelo destino a excepción de su extensión, que en este caso será «*.traces».

- createTraceabilityModel(Collection<Trace>)
  Esta función crea un modelo de trazabilidad propio empleando la información devuelta por la transformación. Este modelo podrá navegarse empleando el editor de trazabilidad diseñado en secciones previas.

El código completo de esta clase se encuentra en el Anexo J.

6.4.3. Interfaz gráfica para la ejecución de transformaciones Plugin es.upv.dsic.issi.qvt.launcher.ui

6.4.3.1. Descripción

El plugin es.upv.dsic.issi.qvt.launcher.ui es el encargado de implementar la interfaz de invocaciones de transformaciones QVT. Esta interfaz se encuentra integrado en el cuadro de diálogo de *ejecución de un programa externo* de Eclipse (disponible en el menú «*Run → Run configurations...*».

Para ello, se ha de implementar un grupo de pestañas gráficas que se integrarán en el cuadro de diálogo de ejecuciones. Este conjunto de pestañas será una instancia de una clase que herede de org.eclipse.debug.ui.AbstractLaunchConfigurationTabGroup (en este caso, la clase QvtLaunchConfigurationTabGroup que se explica más adelante será la que herede de ella). En el listado 6.45 se observa cómo se contribuye esta funcionalidad. En la línea 7 se indica que la clase QvtLaunchConfigurationTabGroup es la que implementará el conjunto de pestañas de la interfaz. Además, tal y como se observa en la línea 9, estas pestañas estarán asociadas al tipo de configuración de ejecución es.upv.dsic.issi.qvt.launcher.launchConfigurationType, que fue definido en la clase QvtLaunchConfiguration del plugin es.upv.dsic.issi.qvt.launcher.

También se observa en la línea 13 que se declara un *shortcut*. En este caso, lo que se hace es asociar un determinado tipo de fichero (los de extensión «*.qvt», línea 39) con este tipo de configuraciones. Para ello se ha de definir una clase que implemente la interfaz org.eclipse.debug.ui.ILaunchShortcut (en este caso será QvtLaunchShortcut). Esta clase implementará los mecanismos para leer el fichero «*.qvt», y rellenar con los datos obtenidos cómo se inicializará con dichos datos la ventana de ejecución de una transformación. Esto será así para distintas perspectivas, que son las que aparecen listadas dentro de la conexión al punto de extensión. El efecto que esta asociación produce es el siguiente: haciendo click derecho sobre un documento con extensión «*.qvt» estará disponible la opción «*QVT Transformation*» dentro de la categoría «*Run as*», tal y como muestra la figura 6.43. Tras ello se abrirá la ventana de ejecución de transformaciones. Si la transformación se ha ejecutado previamente, los datos que aparecerán son los de la última configuración, sino, se rellenará con los datos que pueden conocerse tras analizar el fichero de la transformación.

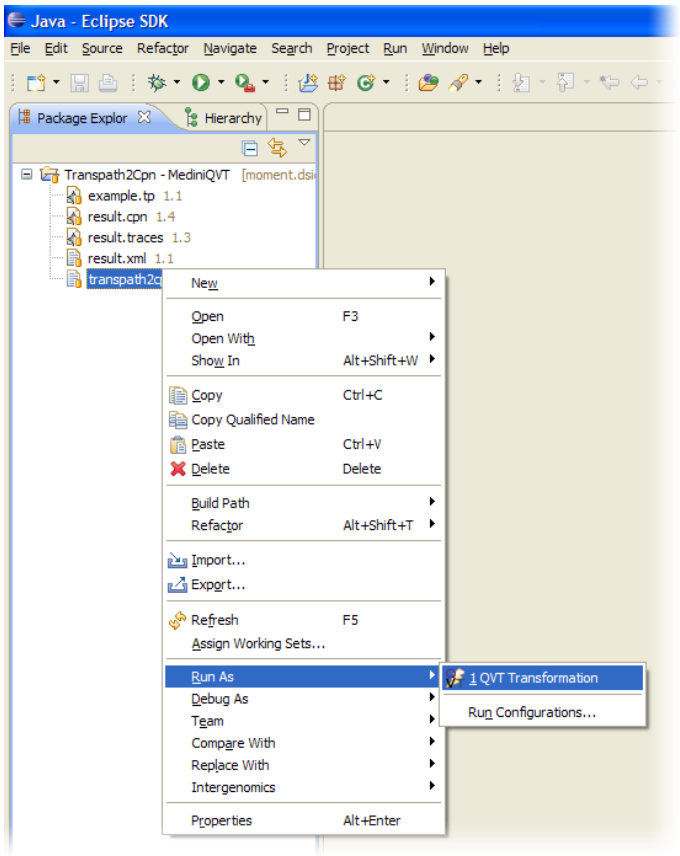


Figura 6.43: Menú «Run as → QVT Transformation»

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         point="org.eclipse.debug.ui.launchConfigurationTabGroups">
      <launchConfigurationTabGroup
            class="es.upv.dsic.issi.qvt.launcher.ui.internal.
                QvtLaunchConfigurationTabGroup"
            id="es.upv.dsic.issi.qvt.launcher.ui.launchConfigurationTabGroup"
            type="es.upv.dsic.issi.qvt.launcher.launchConfigurationType">
      </launchConfigurationTabGroup>
   </extension>
   <extension
         point="org.eclipse.debug.ui.launchShortcuts">
      <shortcut
            class="es.upv.dsic.issi.qvt.launcher.ui.internal.QvtLaunchShortcut"
            icon="icons/transf_invocation.png"
            id="es.upv.dsic.issi.qvt.launcher.ui.shortcut"
            label="QVT Transformation"
```

```
19                  modes="run">
20               <perspective
21                     id="org.eclipse.jdt.ui.JavaPerspective">
22               </perspective>
23               <perspective
24                     id="org.eclipse.jdt.ui.JavaHierarchyPerspective">
25               </perspective>
26               <perspective
27                     id="org.eclipse.jdt.ui.JavaBrowsingPerspective">
28               </perspective>
29               <perspective
30                     id="org.eclipse.debug.ui.DebugPerspective">
31               </perspective>
32               <perspective id="org.eclipse.ui.resourcePerspective"/>
33               <contextualLaunch>
34            <enablement>
35              <with variable="selection">
36               <count value="1"/>
37                <iterate>
38                 <or>
39               <test property="org.eclipse.debug.ui.matchesPattern" value="*.qvt"/>
40                  </or>
41                </iterate>
42               </with>
43            </enablement>
44            </contextualLaunch>
45            </shortcut>
46         </extension>
47      </plugin>
```

Listado 6.45: Archivo de manifiesto plugin.xml de es.upv.dsic.issi.qvt.launcher.ui

6.4.3.2. Dependencias

El plugin tiene las siguientes dependencias:

- org.eclipse.core.runtime

  Este plugin proporciona el entorno básico de ejecución.

- org.eclipse.ui

  Este plugin proporciona constructores básicos (cuadros de diálogos, etc.) para la creación de interfaces gráficas.

- org.eclipse.debug.ui

  El plugin org.eclipse.debug.ui proporciona las clases para tratar con los aspectos gráficos de edición de configuraciones de aplicaciones externas.

- org.eclipse.ui.ide

  Este plugin proporciona algunos cuadros de diálogo avanzados.

- org.eclipse.jface.text

  Este plugin permite acceder al contenido de editores textuales.

- es.upv.dsic.issi.qvt.engine

  El plugin es.upv.dsic.issi.qvt.engine proporciona el motor de ejecución de transformaciones así como las capacidades de análisis y consulta de programas en QVT–Relations.

- es.upv.dsic.issi.qvt.launcher

  Este es el plugin que proporciona el entorno automatizado para la ejecución de configuraciones.

- es.upv.dsic.issi.qvt.launcher.model

  Mediante este plugin es posible definir la configuración completa de la transformación a ejecutar.

6.4.3.3. Descripción de paquetes y clases

Paquete es.upv.dsic.issi.qvt.launcher.ui

Este paquete únicamente contiene la clase activadora del plugin, llamada QvtLauncherUiPlugin. Esta controla el ciclo de vida del plugin así como la funcionalidad visible de éste. Contiene una implementación estándar.

Paquete es.upv.dsic.issi.qvt.launcher.ui.internal

Este es el paquete que implementa la mayor parte de la funcionalidad. A continuación se listan las diferentes clases que contiene así como una breve descripción de cada una.

- ExtensionFilter.java

  Esta clase, que hereda de QvtLauncherUiPlugin.java implementa un filtro para visores estructurados. En este caso, realiza una selección de los elementos que contiene según la extensión. Se emplea para asociarlo a un cuadro de diálogo de selección de archivos.

- QvtLaunchConfigurationTabGroup.java

  Esta clase, que hereda de org.eclipse.debug.ui.AbstractLaunchConfigurationTabGroup implementa un grupo de pestañas (pestañas de tipo org.eclipse.debug.ui.ILaunchConfigurationTab). Para ello se debe implementar el método createTabs(...) que creará las instancias de las pestañas. En este caso este grupo contiene dos tipos de pestañas: la pestaña de configuración de la transformación QvtMainTab y la pestaña común a todos los tipos de ejecuciones org.eclipse.debug.ui.CommonTab.

- QvtMainTab.java

  Esta clase implementa la interfaz gráfica para la configuración de ejecución de transformaciones. Fundamentalmente, para ello se debe implementar el método createControl(...)) heredado de ILaunchConfigurationTab. El contenido de este método es el típico código que hace uso de *SWT* y *JFace* para recrear una interfaz gráfica. Además se deben de implementar los métodos manejadores de los botones «*Apply*», «*Revert*», «*Run*», etc. que permitan salvar la configuración, revertirla, lanzarla, etc. La figura 6.44 muestra su aspecto.

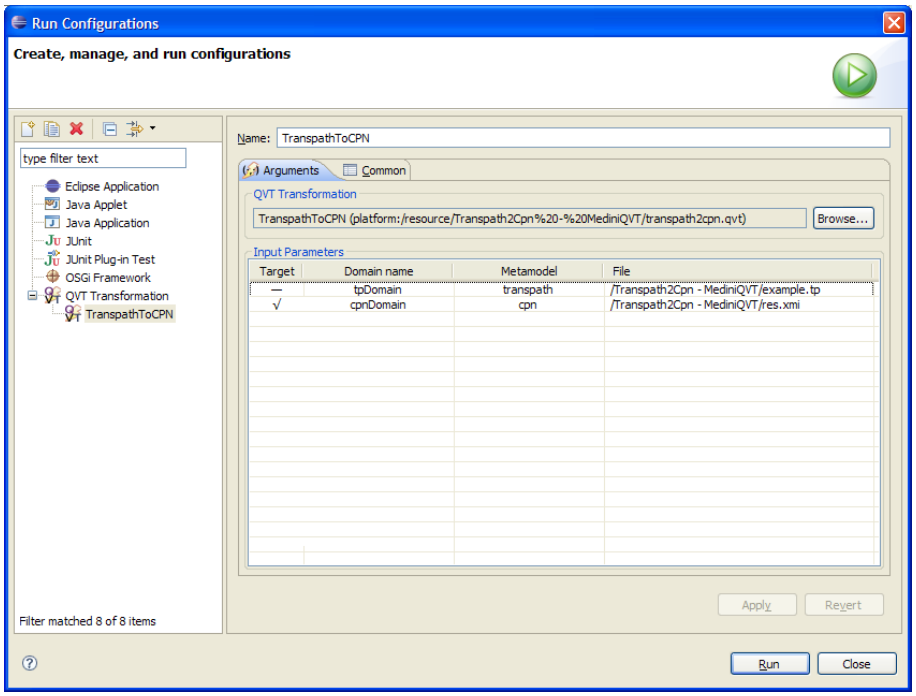


Figura 6.44: Ventana de configuración de la ejecución de una transformación QVT

En este caso, la tabla central hace uso de la API *JFace*, siendo la tabla de tipo org.eclipse.jface.viewers.TableViewer. Esta API sigue el paradigma Modelo/Vista/Controlador, y en este caso para la tabla, necesita de una serie de clases auxiliares que permiten encapsular la funcionalidad de forma sencilla. Estas clases auxiliares se describen a continuación.

- ParamsCellModifier.java

  La clase ParamsCellModifier implementa los mecanismos para modificar los valores del modelo subyacente a la tabla. Además, también indica cuáles de todos los campos serán editables. Para ello se ha de proporcionar la implementación para los métodos canModify(Object, String), getValue(Object, String) y modify(Object, String, Object).

- ParamsContentProvider.java

La clase ParamsContentProvider es la que implementa cuáles son los elementos del modelo que se van a corresponder con cada una de las filas de la tabla. En este caso, el método más significativo es getElements(Object invocation), que en este caso devolverá un vector, donde cada componente será un dominio, es decir, el elemento que se corresponde con cada una de las filas.

- ParamsLabelProvider.java

  La clase ParamsLabelProvider indica cómo se debe consultar el modelo para representar cada una de las columnas de una fila, a partir del elemento del modelo (en este caso, será un elemento de tipo Domain). Para ello se debe de proporcionar fundamentalmente la implementación de los métodos getColumnImage(Object, int), getColumnText(Object, int) e isLabelProperty(Object, String).

- QvtLaunchShortcut.java

  Este clase es la encargada de, dado un fichero con extensión «*.qvt», comprobar que se trata de una transformación válida, y en su caso, intentar rellenar con unos valores adecuados una instancia del modelo de configuración de invocaciones. Esta será la instancia se empleará como posteriormente como valor inicial de la ejecución activa en la clase QvtMainTab. En este caso, la clase QvtLaunchShortcut primero intenta encontrar la ejecución de una configuración previa. En caso de que no exista, se llamará al método QvtinvocationFactory.eINSTANCE.createQvtTransformationInvocation(IFile) plugin es.upv.dsic.issi.qvt.launcher.model para crear una instancia inicial adecuada.

- ResourcesTreeSelectionDialog.java

  Esta clase implementa un cuadro de diálogo de selección de ficheros del *workspace* con alguna funcionalidad añadida personalizada (filtros, ordenación, etc.). El cuadro de diálogo permite especificar tanto ficheros existentes como no existentes, y por ello se emplea para seleccionar tanto los ficheros de los dominios origen, como para especificar la ruta del fichero destino.

Capítulo 7

Manual de MOMENT-QVT

7.1. Descripción

A lo largo de la depuración y uso de la herramienta en la implementación de este trabajo se han identificado numerosas cuestiones que se han de tener en cuenta en el proceso de definición y ejecución de las transformaciones en MOMENT. Por ello, el siguiente capítulo recoge toda la información de interés para el usuario final en el uso de la herramienta. A lo largo de las siguientes secciones se especifica:

- Cuáles son las convenciones de estilo a lo largo del manual de usuario.
- Cómo se instala la herramienta y se prepara para su uso.
- Cómo se deben definir los metamodelos en emplear en EMF, qué restricciones se deben tener en cuenta y cómo se incluyen en la herramienta para poder ser utilizados.
- Cuál es el proceso interno que realiza la herramienta para poder ejecutar las transformaciones, para ayudar en el proceso de depuración de errores.
- Cómo se deben definir las transformaciones teniendo en cuenta convenciones de nombrado, restricciones en las declaraciones (transformaciones, keys, variables, relaciones, expresiones OCL y funciones auxiliares).
- Cuáles son las recomendaciones, patrones y guías de diseño que se deben seguir en el proceso de definición de transformaciones.
- Cómo se emplean las ayudas de la interfaz: creación de una instancia para ser transformada, creación de una transformación, traducción a codigo Maude y gestión de errores.
- Cómo se configura y lanza la ejecución de transformaciones a través de la interfaz de ejecución de MOMENT.

7.2. Convenciones utilizadas.

7.2.1. Terminología empleada.

En el siguiente documento generalmente se referirá a ecore como lenguaje de meta-metamodelado en lugar de cómo lenguaje de modelado (tal y como se emplea generalmente en EMF). Así, los artefactos definidos mediante ecore, serán metamodelos (en lugar de modelos) y los artefactos definidos mediante dichos metamodelos, serán modelos (en lugar de instancias).

No obstante, ha de tenerse en cuenta que, puesto que ecore se define a sí mismo, puede situarse tanto a nivel M3 (como implementación de MOF) como a nivel M2 (como subconjunto del diagrama de clases de UML), pudiendo causar alguna confusión a lo largo del documento.

7.2.2. Convenciones de formato.

En el siguiente documento, los fragmentos de código presentarán el siguiente estilo:

```
1       top relation methodToclassMethod {
2         name1 : String;
3         c1: EClass;   // <-- Declaración
4         i1: Interface;  // <-- Declaración
5
6         checkonly domain umldom op1:EOperation{
7           name = name1,
8           eContainingClass = c1 // <-- Uso
9         };
10        enforce domain javabdom m1:Method{
11          name = name1,
12          owner = i1      // <-- Uso
13        };
14        when{
15          classTointerface(c1,i1);
16        }
17      }
```

Listado 7.1: Ejemplo de fragmento de código

Por otra parte, los párrafos que deban ser especialmente recordados y tenidos en cuenta, se han sombreado ligeramente para resaltarlos sobre el resto del texto, por ejemplo:

> En MOMENT se prohíbe el uso del guión bajo como parte del nombre de los elementos de un metamodelo.

7.3. Instalación de MOMENT.

7.3.1. Requisitos.

7.3.1.1. Maude 2.2 o superior.

MOMENT incluye el kit de desarrollo Maude Development Tools [73]. Esto implica que se necesita tener Maude [70] (version 2.2 o superior) instalado en el equipo.

Para ejecutar MOMENT en Windows, basta con emplear el instalador Maude for Windows [74].

En caso de que se desee instalar en un equipo Linux, deben seguirse las instrucciones de [71].

7.3.2. Instalación.

En el directorio de releases del ftp de MOMENT [72] se suministran todos los ficheros necesarios para la instalación de MOMENT[1]. Estos son:

1. Archivo «eclipse-SDK-3.2.1-win32.zip» para Windows.

2. Archivo «MOMENT Update Site.zip».

3. Archivo «MOMENT Manual *número_versión*.pdf».

El usuario únicamente deberá descargar estos archivos e iniciar eclipse de la manera habitual, siendo el proceso de instalación similar para todas las plataformas. La instalación del prototipo MOMENT se llevará a cabo desde el menú «Help → Software Updates → Find and Install».

[1]En el FTP sólo se incluye la versión de eclipse para Windows. En Linux o MacOSX el proceso de instalación es similar, únicamente sería necesario descargar el correspondiente fichero «eclipse-SDK-3.2.1-*plataforma*.*ext*» para la plataforma deseada desde [67].

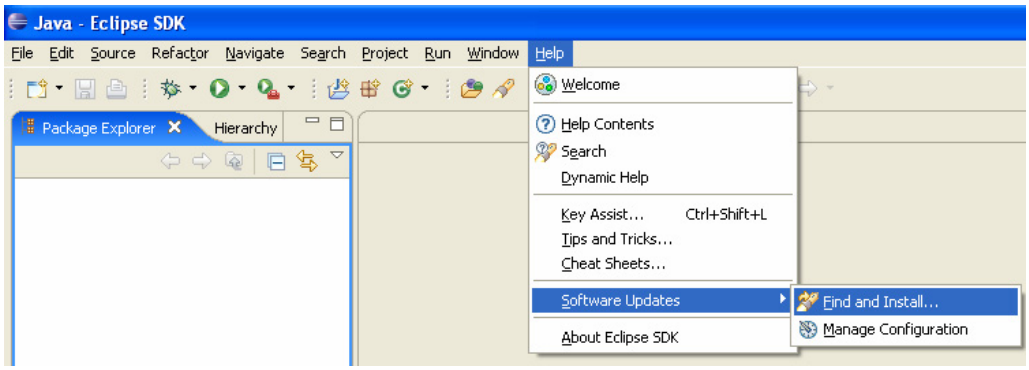


Figura 7.1: Asistente de Instalación.

Seguidamente se deberá seguir el asistente de instalación propio de Eclipse. En la ventana «Install/Update» seleccionaremos «Search for new features to install».

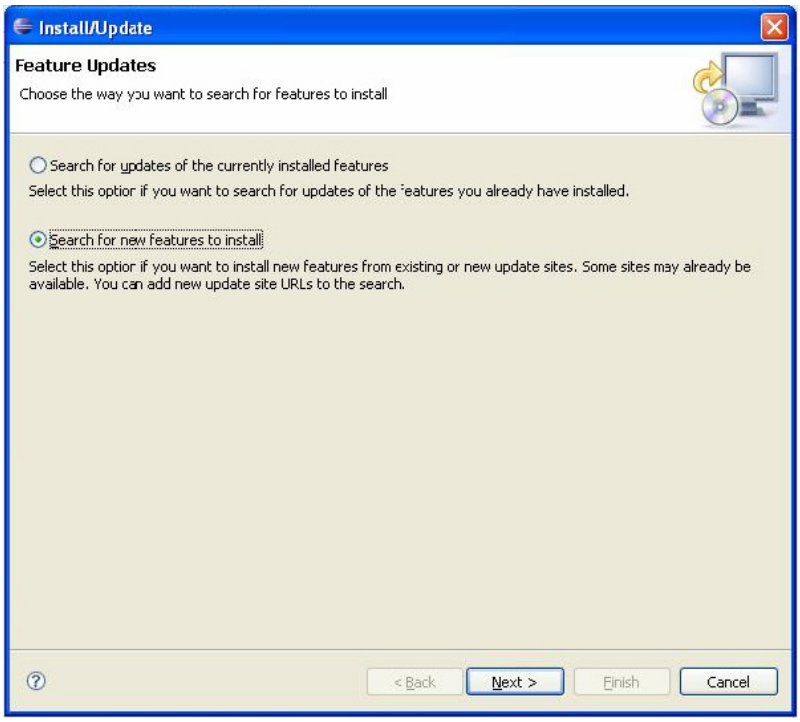


Figura 7.2: Asistente de Instalación.

Posteriormente desde la ventana «Install», pulsaremos el botón «New Archived Site» y seleccionaremos el fichero «MOMENT Update Site.zip» descargado.

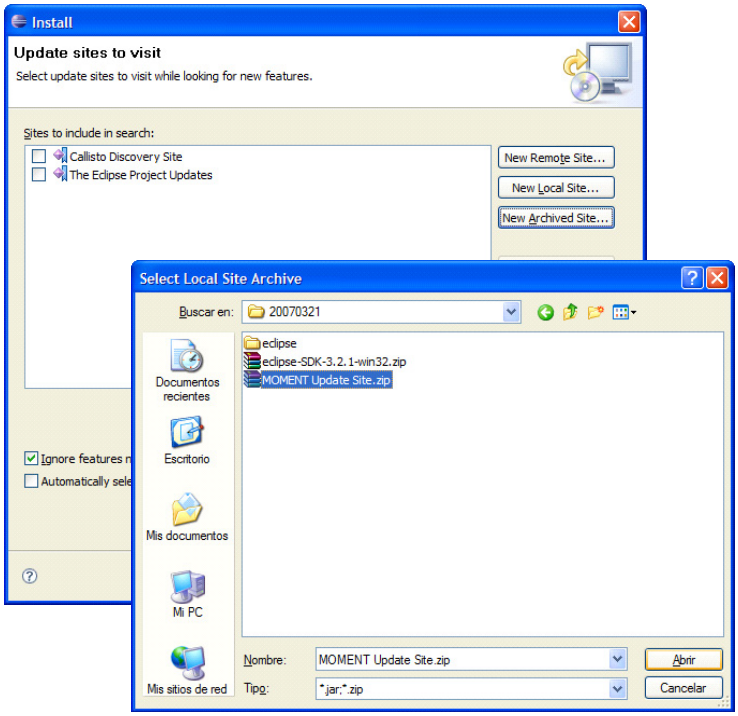


Figura 7.3: Selección del Update Site.

Una vez seleccionado el fichero comprimido de los plugins de MOMENT, podremos cambiar, si lo deseamos, el nombre del update site. Sino, podemos apretar directamente el botón «OK».

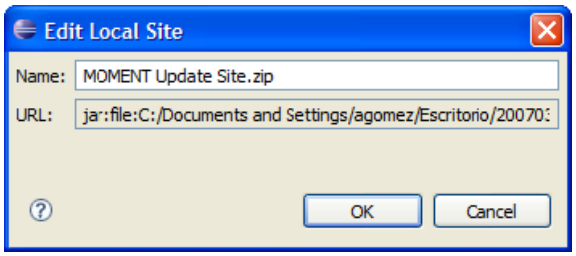


Figura 7.4: Introducción del Update Site.

En este punto, el sitio se habrá añadido a la lista de sitios que aparecen por defecto.

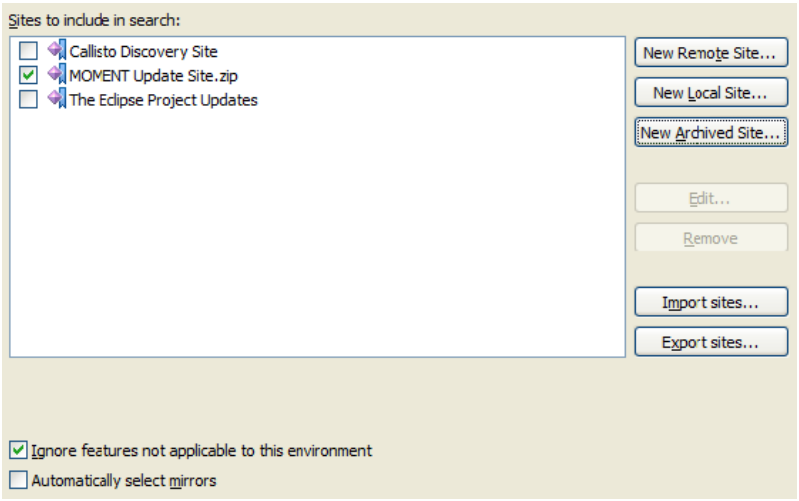


Figura 7.5: Selección del Update Site.

Tras finalizar la configuración del Update Site, pulsaremos el botón «OK» asegurándose de que dentro del área «Sites to include in search» aparece seleccionado el Update Site que se acaba de incluir. La casilla de verificación creada con la nueva entrada en el Update Site Manager debe estar seleccionada. A continuación pulsaremos el botón «Finish» para proceder a la instalación.

La primera ventana emergente mostrará las *features* (según la terminología Eclipse) encontradas en la instalación de los plugins del prototipo, agrupadas en 3 secciones:

1. ANTLR: contiene los plugins que contienen las librerías de ANTLR, una utilidad para creación de analizadores y compiladores.
2. Other Dependencies: contiene los plugins que dan soporte al framework de modelado EMF.
3. MOMENT: contiene los plugins del MOMENT.

Es necesario seleccionar todas y cada una de las *features* incluidas en cada una de los grupos y pulsaremos el botón Next.

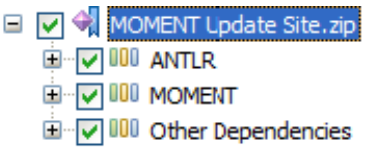


Figura 7.6: Asistente de instalación.

La siguiente pantalla mostrará las licencias de cada una de las *features* que se van a instalar. Tras aceptar las condiciones de la licencia seleccionaremos el botón de radio «I accept the terms in the licesement agreement» y pulsaremos el botón *Next*.

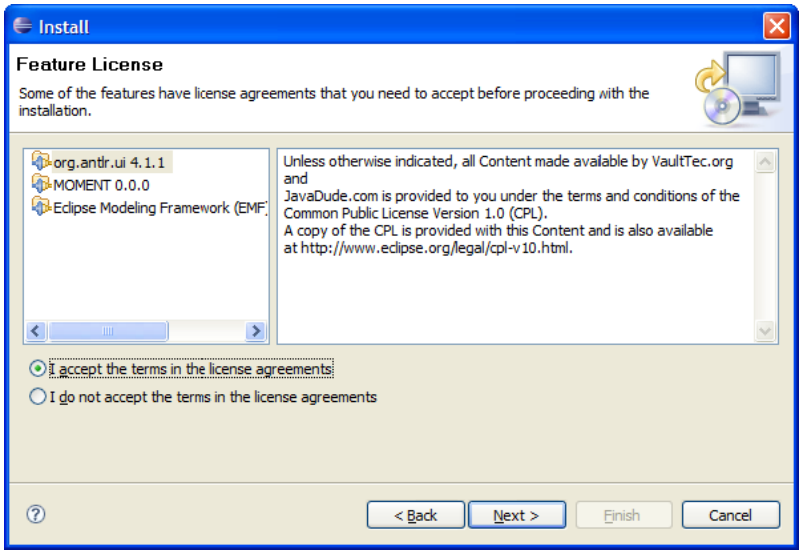


Figura 7.7: Licencias.

El paso siguiente, nos solicitará dónde deseamos instalar los plugins. Se recomienda la opción por defecto.

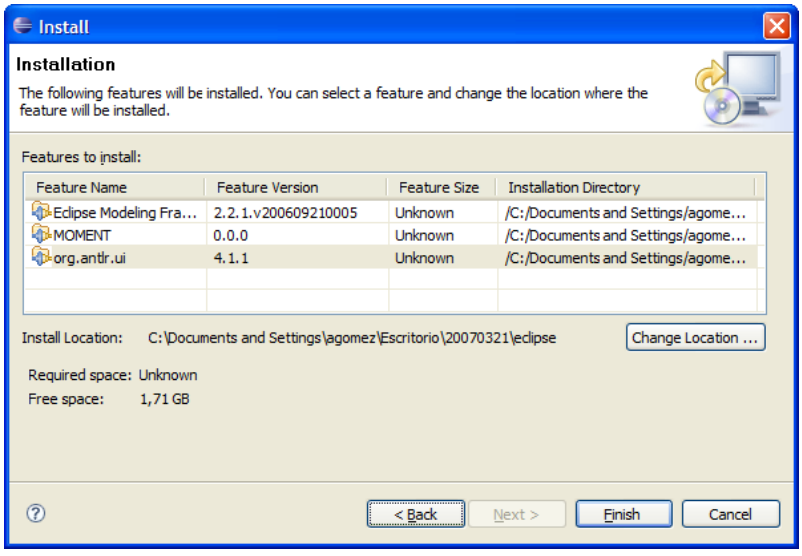


Figura 7.8: Selección del directorio de instalación de los plugins.

A continuación se mostrará una ventana en la que pide la confirmación de la instalación de cada una de las features en las que están agrupados los plugins.

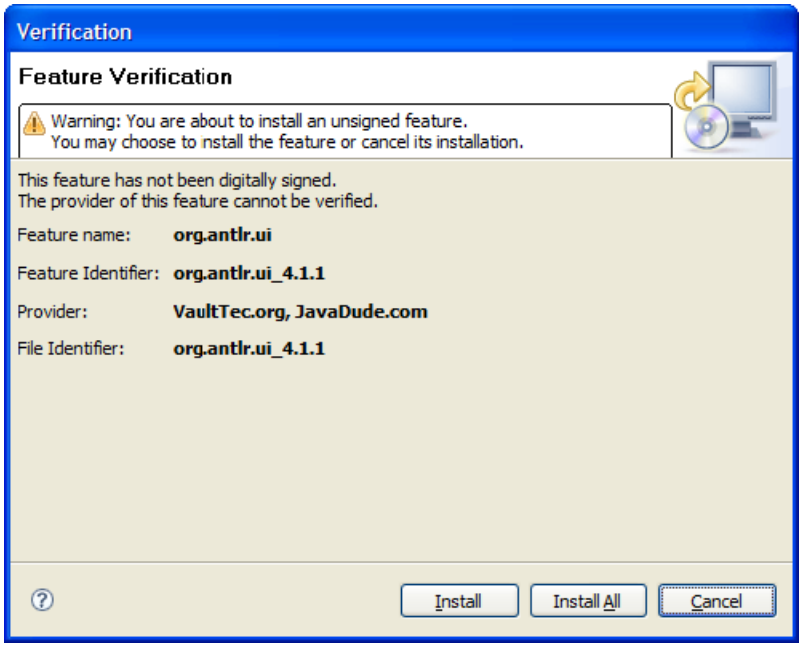


Figura 7.9: Verificación de la instalación.

Una vez instalados los plugins, será necesario reiniciar Eclipse si quiera utilizar las funcionalidades suministradas.

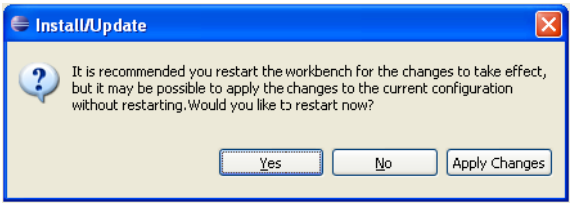


Figura 7.10: Fin de la instalación.

7.3.3. Configuración de MOMENT.

El único paso previo que se debe realizar para poder emplear MOMENT es la configuración del kit de desarrollo Maude Development Tools. Para ello, deberemos indicar mediante la ventana de preferencias de Eclipse, en la sección Maude, las rutas del fichero ejecutable de Maude y el fichero con la especificación de Full Maude.

Para mayor detalle acerca de las Maude Development Tools, puede consultarse el capítulo «Maude Development Tools» de la ayuda en línea de Eclipse —accesible a través del menú «Help → Help Contents»— una vez se ha instalado MOMENT.

En el apartado 7.9 se muestra cómo utilizar MOMENT para ejecutar las transformaciones de ejemplo que se proporcionan.

7.4. Definición de metamodelos.

7.4.1. Nombrado de clases y atributos.

7.4.1.1. Identificadores válidos.

Identificador[2]: [a-zA-Z]([a-zA-Z0-9])*

> En MOMENT se prohíbe el uso del guión bajo como parte del nombre de los elementos de un metamodelo.

A esta restricción, se le añade la restricción impuesta por Java. En EMF se desaconseja emplear también aquellos caracteres que no sean válidos en Java (ya que los metamodelos pueden ser proyectados a código).

7.4.1.2. Restricciones de nombrado.

Actualmente existen restricciones en el nombrado de atributos. Se desaconseja encarecidamente repetir el nombre de los atributos en diferentes clases ya que pueden producirse conflictos en los modelos de trazabilidad, pudiendo contener información incorrecta. En caso de que esto sea inevitable, en ningún caso los atributos con nombre repetido deberán ser de tipos distintos (o si son referencias, deben apuntar a la misma clase, p.e. atributo *owner* del metamodelo rdbms).

7.4.2. Metamodelos contenidos en diferentes ficheros ecore.

Los metamodelos empleados en MOMENT deben ser autocontenidos. Esto es, a excepción de referencias a los tipos de datos de ecore (EString, EBoolean, etc.), no deben contener referencias a ningún elemento de un fichero externo.

[2]Según la sintaxis de expresiones regulares de java[61].

7.4.3. Generación de modelos ecore a partir de ficheros XSD.

La generación de modelos ecore a partir de esquemas XSD no se soporta. No se soportan los elementos generados de forma automática EFeatureMapEntry y EStringToStringMapEntry.

Los términos generados a partir de modelos que conforman a metamodelos obtenidos a partir de esquemas XSD (en el espacio tecnológico EMF) pueden no hacer pattern-matching con las correspondientes teorías generadas a partir de sus metamodelos (en el espacio tecnológico Maude).

7.4.4. Registro de metamodelos.

Para trabajar con modelos que conformen a un metamodelo definido mediante ecore, éste debe encontrarse registrado en EMF. Se puede comprobar cuáles son los metamodelos actualmente cargados en el registro de EMF mediente la vista Metamodels (Window → Show view → Other... → MOMENT → Metamodels).

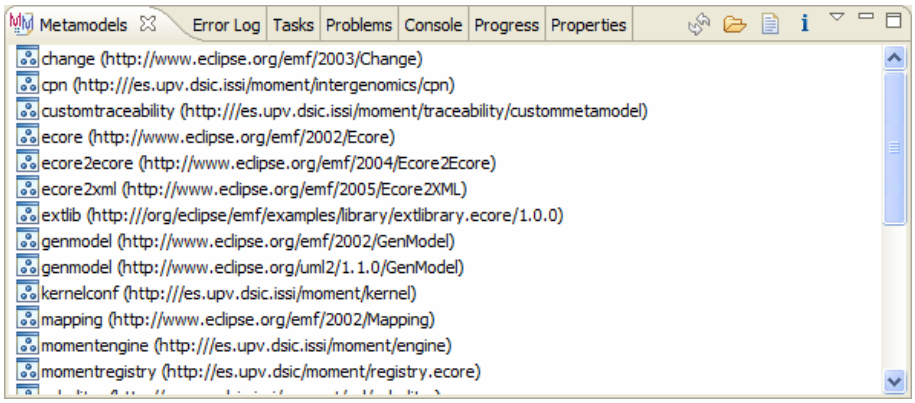


Figura 7.11: Vista de metamodelos registrados.

Para registrar un metamodelo en EMF, se pueden realizar (alternativamente) cualquiera de las siguientes opciones:

7.4.4.1. Creación de un plug-in EMF.

Este es el mecanismo por defecto de EMF para la instalación de un metamodelo ecore en Eclipse. Se deberá emplear el asistente para la creación de un nuevo plugin de

EMF a partir de un modelo ecore.

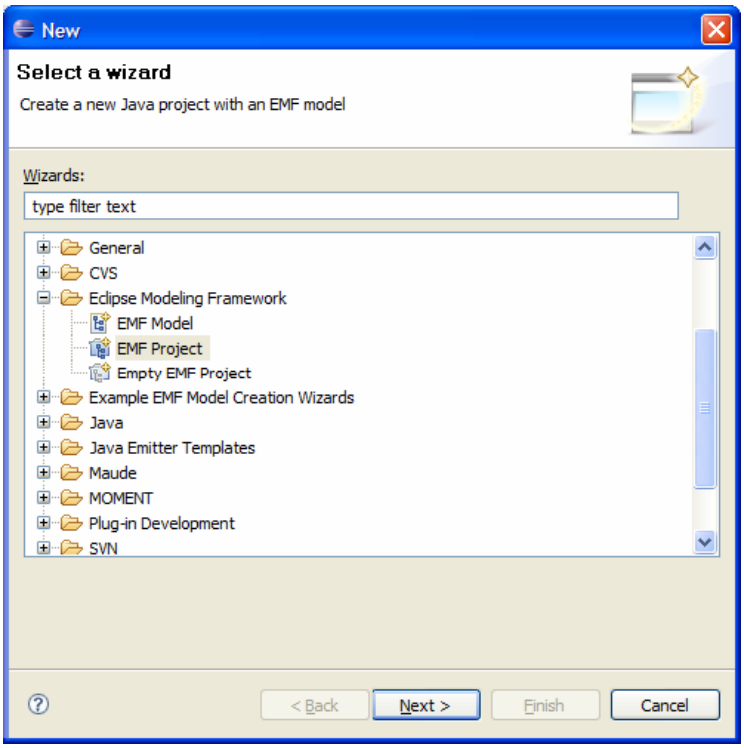


Figura 7.12: Selección del asistente de creación de un proyecto EMF.

Una vez creado el proyecto, se deberá generar el código del metamodelo, (y si se desea del editor y asistentes) y deberá instalarse posteriormente. Para más información acerca de este método consúltese [66].

> Si el plugin se compila e instala el metamodelo estará siempre disponible para ser usado. Como contrapartida tenemos que los plug-ins deberán ser regenerados, recompilados y reinstalados cada vez que se realicen variaciones sobre el metamodelo.

7.4.4.2. Registro manual.

Para evitar tener que regenerar los plug-ins cada vez que se realizan modificaciones sobre los metamodelos, la herramienta MOMENT proporciona un atajo para registrar los metamodelos en EMF directamente, a partir del fichero «*.ecore». Para ello, se deberá hacer

clic derecho sobre el fichero *.ecore, y se seleccionará la opción «MOMENT Extensions → Register model».

Cada vez que se realicen variaciones sobre el metamodelo se puede repetir este proceso para registrar la nueva versión[3],[4].

> Los metamodelos que se registren de esta manera únicamente estarán disponibles mientras la sesión de eclipse no se cierre. Al iniciar de nuevo Eclipse, se deberán registrar de nuevo.

7.5. Proceso de ejecución de transformaciones QVT en MOMENT.

El proceso de transformación que se lleva a cabo en MOMENT consta de tres fases: análisis, proyección a Maude y ejecución y proyección a EMF (figura 7.13). Todas ellas quedan ocultas al usuario, siendo únicamente necesaria la especificación de la transformación por parte del mismo.

Para llevar a cabo estas tres fases, son necesarios un conjunto de componentes funcionales que forman la arquitectura de MOMENT.

1. QVT Parser: encargado de analizar un programa QVT Relations de entrada y generar su modelo correspondiente. Este componente interviente en la fase de análisis.

   Esta etapa comienza con un análisis léxico y sintáctico del programa QVT Relations. Para este análisis es necesario consultar los metamodelos respectivos a los modelos participantes en la transformación. Por consiguiente, es prerrequisito imprescindible que estos metamodelos estén registrados en EMF.

2. MOMENT Relations: encargado de generar y proyectar código Maude a partir de la especificación de una transformación.

3. OCL Parser: encargado de analizar expresiones OCL y generar su código Maude correspondiente.

[3] En este caso, si el metamodelo ya estaba registrado, la vista de metamodelos puede contener entrada repetidas.

[4] El parser de programas QVT puede que no tenga en cuenta los cambios realizados sobre los metamodelos modificados si una versión anterior intervenía en un programa analizado anteriormente. Esto provocará que el modelo «*.qvt» obtenido no esté actualizado. En este caso, deberá reiniciarse eclipse.

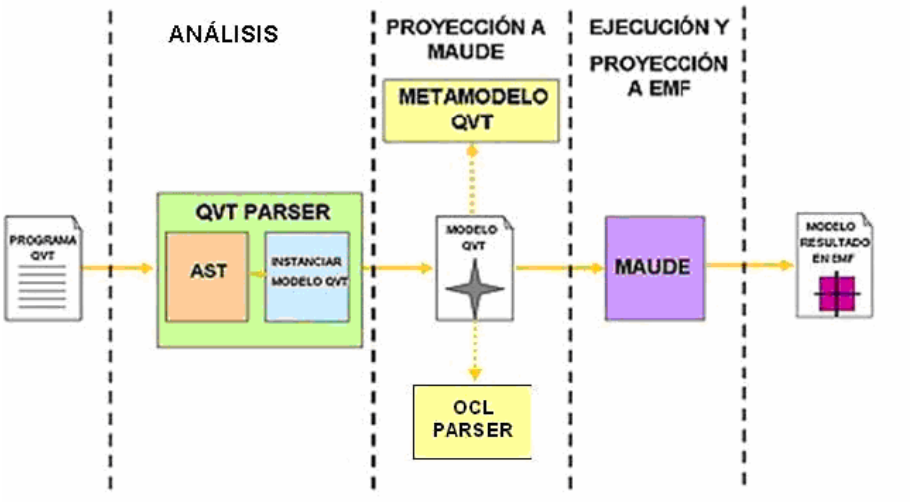


Figura 7.13: Proceso de ejecución de un programa QVT Relations en MOMENT.

7.5.1. Fase de análisis

Esta etapa comienza con un análisis léxico y sintáctico del programa QVT Relations. Para este análisis es necesario consultar los metamodelos respectivos a los modelos participantes en la transformación. Por consiguiente, es prerrequisito imprescindible que estos metamodelos estén registrados en EMF.

7.5.2. Ejecución y proyección a EMF

Llegado este punto, se dispone en el espacio tecnológico Maude de toda la información necesaria para lanzar la transformación a ejecución. Desde Maude, se aprovecha el mecanismo de reducción modulo las ecuaciones generadas a partir de un programa QVT Relations, como brevemente se indica en el apartado anterior y de pattern-matching, obteniendo el término resultante que en el ejemplo anterior se corresponde con un esquema relacional. Finalmente, este término es proyectado de vuelta a EMF utilizando los puentes definidos a nivel M1 entre ambos espacios tecnológicos. Se pueden encontrar más detalles del proceso de compilación a código Maude en [6].

7.6. Definición de transformaciones.

7.6.1. Convenciones de nombrado.

Las convenciones de nombrado son diferentes según el ámbito (variables, reglas, helpers, etc). En general, a fin de evitar problemas, se recomienda seguir las restricciones generales de nombrado aplicables a metamodelos.

Identificador[5]: [a-zA-Z]([a-zA-Z0-9])*

7.6.1.1. Identificador de las reglas o relations

> El identificador de las reglas o relations no puede contener el carácter guión « - » o guión bajo « _ ».

7.6.2. Declaración de la transformación.

En una transformación, se indicarán los metamodelos de los diferentes dominios mediante su correspondiente URI en EMF.

Habrá 1 único modelo destino especificado a través de la palabra reservada target.

Cuando la URI asociada a alguno de los modelos participantes en la transformación contenga dos o más barras seguidas "//" colocaremos el carácter de escape "\" para que no se confunda con el token que especifica un comentario de línea.

```
1  transformation umlToRdbms (
2    ecoreDomain:http:\/\/www.eclipse.org/emf/2002/Ecore,
3    target rdbmsDomain:http:\/\/\/es.upv.dsic.issi/moment/rdbms
4  )
```

Listado 7.2: Declaración de la transformación umlToRdbms

Existirán tantos modelos origen como sea necesario para definir la transformación. Eso sí, habrá un único modelo director, y éste será el primero en aparecer. En la siguiente definición el modelo director será el identificado por el alias ecoreDomain1.

```
1  transformation umlToRdbms(
2    ecoreDomain1:http:\/\/www.eclipse.org/emf/2002/Ecore,
3    ecoreDomain2:http:\/\/www.eclipse.org/emf/2002/Ecore,
4    ecoreDomain3:http:\/\/www.eclipse.org/emf/2002/Ecore,
```

[5]Según la sintaxis de expresiones regulares de java[61]

```
5         target rdbmsDomain:http:\/\/\/es.upv.dsic.issi/moment/rdbms
6       )
```

Listado 7.3: Ejemplo de transformación con varios dominios origen

7.6.3. Declaración de keys.

> Siguiendo el estándar QVT, para toda clase del metamodelo destino se debe indicar cuáles son los atributos que identifican unívocamente a cada objeto de dicha clase.

```
1       key Schema {name};
2       key Table {schema,name};
3       key Column {owner,name};
4       key ForeignKey {owner,name};
```

Listado 7.4: Declaración de keys en QVT-Relations

Se declararán por orden de dependencia. Por ejemplo el término Column (que se identifica por su nombre y por su owner —no es otro que la Table-), debe ser declarado con posterioridad al término Table.

Hay que asegurarse de que los atributos y/o relaciones que forman parte de la Key formarán parte del elemento creado en la regla de transformación que se encargue de ello.

> No se puede declarar una key de un elemento que no se ha definido o utilizado en ninguna de las reglas de transformación.

7.6.4. Declaración de variables

Cualquier variable de tipo simple deberá ser declarada con anterioridad a su uso.

Toda variable estructurada dentro de una regla puede ser definida al vuelo, o mediante una declaración previa al inicio de la relation.

Sin embargo, nunca deberán aparecer ambos tipos de declaración juntos, ya que de momento el parser las duplica tanto en la declaración como en la llamada, además de contener en algunos casos diferente información.

7.6.4.1. Declaración Previa

```
top relation methodToclassMethod {
  name1 : String;
  c1: EClass;   // <-- Declaración
  i1: Interface;  // <-- Declaración

  checkonly domain umldom op1:EOperation{
    name = name1,
    eContainingClass = c1 // <-- Uso
  };
  enforce domain javabdom m1:Method{
    name = name1,
    owner = i1      // <-- Uso
  };
  when{
    classTointerface(c1,i1);
  }
}
```

Listado 7.5: Declaración de variables previa

7.6.4.2. Declaración al Vuelo

```
top relation methodToclassMethod {

  checkonly domain umldom op1:EOperation{
    name = name1,
    eContainingClass = c1: EClass{} // <-- Declaración al vuelo
  };

  enforce domain javabdom m1:Method{
    name = name1,
    owner = i1: Interface{},    // <-- Declaración al vuelo
  };
  When {
    classTointerface(c1,i1);
  }
}
```

Listado 7.6: Declaración de variables al vuelo.

7.6.4.3. Restricciones en la declaración de variables.

En una regla o relation no puede haber variables cuyo identificador sea subcadena de un identificador de otra variable. Por ello es aconsejable, como medida de precaución, introducir un sufijo numérico en el identificador de toda variable.

Ejemplo:

```
relation xxxx {

  attributeName1, ecoreTypeName1 : String;

  checkonly domain ecoreDomain1 a1:EAttribute {
    name = attributeName1,
    eType = ecoretype : EDataType {
      name = ecoreTypeName1
    }
  };
}
```

Listado 7.7: Recomendaciones en la declaración de variables.

7.6.5. Declaración de relations

7.6.5.1. Declaración de dominios en una regla o relation

Obligatoriamente todas las relations deberán tener al menos tantos dominios checkonly como numero de modelos origen participen en la transformación, aunque se pueden definir más.

7.6.5.2. Restricciones y precondiciones

Se pueden utilizar funciones para expresar restricciones o precondiciones en las reglas. Para ello, las funciones deberán devolver un valor booleano y deberán ser llamados desde el bloque when de la regla.

```
relation PrimitiveAttributeToColumneAttributes {
  ecoreTypeName : String;

  XXXX
  XXXX
  XXXX

  when {
    IsPrimitiveDatatype(ecoreTypeName);
  }
} //end relation

function IsPrimitiveDatatype(datatype: String):Bool
{
((datatype = 'EInt') or (datatype = 'EBoolean') or
  (datatype = 'EString') or (datatype = 'EDate'))
```

```
17      }
```

Listado 7.8: Precondiciones en reglas.

7.6.5.3. Obteniendo un elemento no disponible en el dominio de una regla.

También podemos utilizar las functions para obtener un término concreto, no disponible en los dominios origen de la relation y que es necesario para la construcción del término resultante en una regla concreta.

Las llamadas a otras reglas del bloque WHEN tendrán como último parámetro el resultado de la regla o helper.

Ejemplo 1:

```
1       relation 1 {
2
3         k1 : key;
4
5         checkonly domain xx s1:Schema{
6           name = n1,
7           table = t1:Table{}
8         };
9
10        enforce domain xx: t2:Table{
11          key = k1
12        }
13
14        when {
15          ObtainReferredPrimaryKey(t1,k1));
16          // Se pasa como último parámetro, la variable
17          // "k1" que realizará patternMatching con el
18          // resultado de "ObtainReferredPrimaryKey" y
19          // que se utilizará y formará parte del término
20          // resultante de la regla (bloque enforce domain)
21        }
22      }
23
24      function ObtainReferredPrimaryKey(table: Table):Key {
25          table.key
26      }
```

Listado 7.9: Obteniendo un elemento no disponible en el dominio de una regla (ejemplo 1).

Ejemplo 2:

```
1       relation 2 {
2         propVar1 : Variable;
3
4         checkonly domain xx obj1:ObjectVariable{};
```

```
5
6         checkonly domain yyy objContSource1:ObjecVariable{};
7
8         // .........
9
10        when {
11          getVariablesContainer(obj1, objContSource1, propVar1);
12          // Se pasa como último parámetro , la
13          // variable  "propVar1" que realizará
14          // patternMatching con el resultado de
15          // "getVariablesContainer" y que posteriormente
16          // utilizaremos en la llamada a la siguiente
17          // regla a ejecutar (bloque WHERE)
18          // "VariableNestedToVariableFlattened"
19        }
20
21        where {
22          VariableNestedToVariableFlattened(propVar1);
23        }
24      }
25
26      function getVariablesContainer ( obj1:ObjectTemplateExp ,
27             obj2:ObjectTemplateExp) : String {
28
29        (
30          (obj1.propertyTemplateExpresion.objContainer.
31            partObjTemplateExp.valueOclProperty.
32            variableOCLExp ->asOrderedSet()->
33            intersection(
34              obj2.partObjTemplateExp.valueOclProperty.
35              variableOCLExp->asOrderedSet()
36            )
37          ) -> first().name
38        )
39
40      } //end function  getVariablesContainer
```

Listado 7.10: Obteniendo un elemento no disponible en el dominio de una regla (ejemplo 2).

7.6.5.4. Obteniendo elementos devueltos por un bloque WHEN.

Los elementos creados a través de una llamada when no forman parte del conjunto de términos resultante en la transformación, solo se obtiene su identificador para hacer referencias. Si queremos añadir además el término conseguido en una llamada del bloque when (por medio de su referencia) en el conjunto resultante de términos de la transformación será necesario además duplicar la llamada en el bloque where.

7.6.5.5. Invocaciones en el bloque where.

Las llamadas en el bloque where pueden no contener el parámetro que hace pattern matching con el resultado.

```
relation Transformation2Transformation {

  checkonly domain xxx{};
  enforce domain yyy{};

  when {
    TPackage2TPackage(p1,p2);
    TModel2TModel(sm1,sm2);
    TModel2TModel(tm1,tm2);
  }

  where {
    TModel2TModel(sm1);
    TModel2TModel(tm1);
  }
}
```

Listado 7.11: Invocación de reglas en el bloque where.

7.6.6. Expresiones OCL.

7.6.6.1. Desambigüación de las expresiones OCL.

Para que las expresiones OCL sean traducidas correctamente en una expresión MAUDE, es preciso que se haga uso de paréntesis para diferenciar las distintas operaciones existentes en una misma expresión.

```
(((obj1.propertyTemplateExpresion.objContainer.partObjTemplateExp.valueOclProperty.
    variableOCLExp) ->asSet())->intersection(( obj2.partObjTemplateExp.
    valueOclProperty.variableOCLExp)->asSet()) -> isEmpty())
```

Listado 7.12: Desambiguación de expresiones OCL.

7.6.6.2. Construcción de expresiones OCL.

En una expresión OCL todas las variables deben pertenecer al mismo metamodelo. El parser OCL únicamente trabaja con un único contexto del que parte la navegación.

7.6.6.3. Uso de expresiones OCL en una regla.

> En general, cuando se empleen expresiones OCL a la parte derecha de una asignación a un atributo/referencia, se recomienda hacerlo mediante una llamada a función.

Ejemplo:

```
1   relation ComplexMoleculeToProduct {
2
3     checkonly domain tpDomain molec1 : Molecule {
4       statesOf = simplemolec1 : Molecule {}
5     };
6
7     checkonly domain cpnDomain gb1 : Globbox {};
8
9     checkonly domain cpnDomain block1 : Block {};
10
11    enforce domain cpnDomain prod1 : Product {
12      block = block1,
13      idname = GetMoleculeType(molec1)
14    };
15
16    where {
17      SimpleMoleculeToEnumerated(simplemolec1, gb1, prod1);
18    }
19  }
20  // ...
21  function GetMoleculeType(molec : Molecule) : String
22  {
23    if (molec.statesOf -> isEmpty())
24    then
25      if (molec.klass -> includes('adaptor proteins'))
26      then 'A'
27      else
28        if (molec.klass -> includes('receptors'))
29        then 'R'
30        else 'O'
31        endif
32      endif
33    else
34      if (molec.name = 'LPS:LBP')
35      then 'OA'
36      else
37        if (molec.name = 'LPS:LBP:CD14')
38        then 'OAR'
39        else
40          if (molec.name = 'ST2:TIRAP'
41            or molec.name = 'ST2:MyD88')
```

```
42              then 'RA'
43              else 'Unknown' // No debería llegar aquí
44              endif
45            endif
46          endif
47        endif
48      }
```

Listado 7.13: Uso de expresiones OCL en una regla.

7.6.7. Declaración de Funciones.

7.6.7.1. Cuerpo de las funciones.

> El cuerpo de una función puede ser únicamente una expresión OCL.

7.6.7.2. Invocación de una función desde otra función.

> De momento, no se permite las invocaciones entre funciones.

7.7. Guías para la definición de transformaciones.

7.7.1. Definición de los metamodelos.

> En general, por cómo se realiza el proceso de transformación, se recomienda que todas las asociaciones que no sean de aridad máxima 1 en ambos lados, sean navegables en ambas direcciones (al menos para los metamodelos destino).

Por ejemplo, suponiendo que para una asignatura podemos tener asociados n documentos de examen, pero dichos documentos únicamente están asociados a dicha asignatura la asociación deberá ser navegable en ambos sentidos:

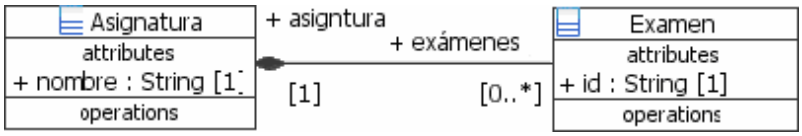


Figura 7.14: Ejemplo de composición [1] — [0..*].

7.7.2. Proceso de ejecución de una transformación.

Suponiendo que el caso anterior es parte del metamodelo destino, para definir las reglas que deben generar una asignatura y el conjunto de exámenes asociados a ella, la transformación se debe ejecutar de la siguiente manera:

1. En primer lugar, se debe definir la regla que cree el objeto asignatura.
2. En segundo lugar, se debe llamar a la regla que cree un examen.
3. Una vez se ha creado el examen, se debe rellenar el rol «asignatura» de la clase «Examen». La referencia inversa (de asignatura a examen) se completa de forma automática al obtener el modelo resultado.

En el siguiente ejemplo, ilustramos la forma de proceder. Los metamodelos origen y destino tienen la misma estructura, a diferencia de que a las clases del metamodelo origen se les ha añadido el sufijo «Src» para evitar confusiones.

```
1       relation AsignaturaSrcToAsignaturaDst {
2
3         asignaturaNombre1 : String;
4
5         checkonly domain srcDomain srcAsig1 : AsignaturaSrc {
6           nombre = asignaturaNombre1,
7           examenes = srcExam1 : DocumentoExamenSrc {}
8         };
9
10        enforce domain dstDomain dstAsig1 : Asignatura {
11          nombre = asignaturaNombre1
12        };
13
14        where {
15          ExamenesSrcToExamenesDst(srcExam1, dstAsig1);
16        }
17      }
18
19      relation ExamenesSrcToExamenesDst {
20
21        idExam1 : String;
22
```

```
23          checkonly domain srcDomain srcExam1 : DocumentoExamenSrc {
24            id = idExam1
25          };
26
27          checkonly domain dstDomain dstAsig1: Asignatura {
28          };
29
30          enforce domain dstDomain dstExam1 : DocumentoExamen {
31            id  = idExam1,
32            asignatura = dstAsig1
33          };
34        }
```

Listado 7.14: Ejemplo de regla para la creación de elementos en asociaciones (1) – (0..*).

7.7.3. Transformaciones de relaciones n–n.

MOMENT soporta transformaciones de aridad mayor que uno a ambos lados. A continuación mostraremos un ejemplo de cómo se realiza el proceso de transformación en estos casos.

En MOMENT, la aplicación de una regla puede modificar el estado de un objeto previamente creado. En este caso, si el objeto previo contenía referencias a otros objetos que habían sido calculadas con anterioridad, la aplicación de otra regla añadirá los nuevos valores a los ya existentes.Para determinar si un objeto que se va a crear existe previamente se hace uso de los elementos declarados como «key» en la transformación.

Un ejemplo de este caso podría ser un modelo que almacene libros y autores (un autor puede haber escrito n libros, y un libro puede estar escrito por n autores).

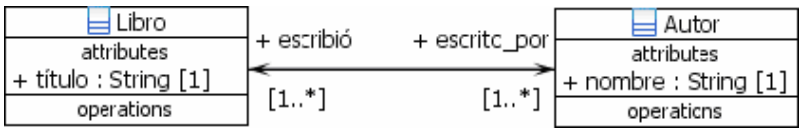


Figura 7.15: Asociación de aridad mayor que uno a ambos lados.

Suponiendo que tenemos dos autores que han participado en escribir dos libros, de la siguiente manera:

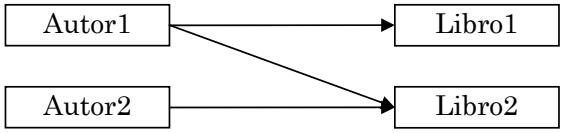


Figura 7.16: Escenario de ejemplo para transformaciones de asociaciones n–n.

Podría darse el siguiente escenario en el proceso de creación del modelo resultado:

1. Primero se crea el primer autor, con los libros en los que participa

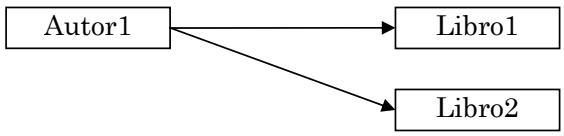


Figura 7.17: Escenario de ejemplo para transformaciones de asociaciones n–n.

2. Posteriormente se crea el segundo autor.

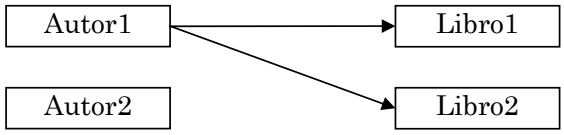


Figura 7.18: Escenario de ejemplo para transformaciones de asociaciones n—n. Paso 2.

3. Por ultimo se procede a la creación del libro en el que participa el Autor2. Al crear de nuevo este libro, se tiene en cuenta el valor previo del rol «escrito_por», añadiendo la nueva referencia entre Autor2 y Libro2:

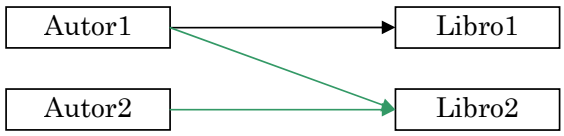


Figura 7.19: Escenario de ejemplo para transformaciones de asociaciones n—n. Paso 3.

7.8. El editor de QVT textual.

7.8.1. Creación de una nueva transformación.

Para crear una nueva transformación definida mediante el lenguaje QVT-Relations textual, se deberá crear un nuevo fichero con extensión «*.qvtext».

Para ello, se puede seleccionar la opción «File», de la carpeta «General», del menú de asistentes de Eclipse —que aparece mediante el atajo de teclado CTRL+N—...

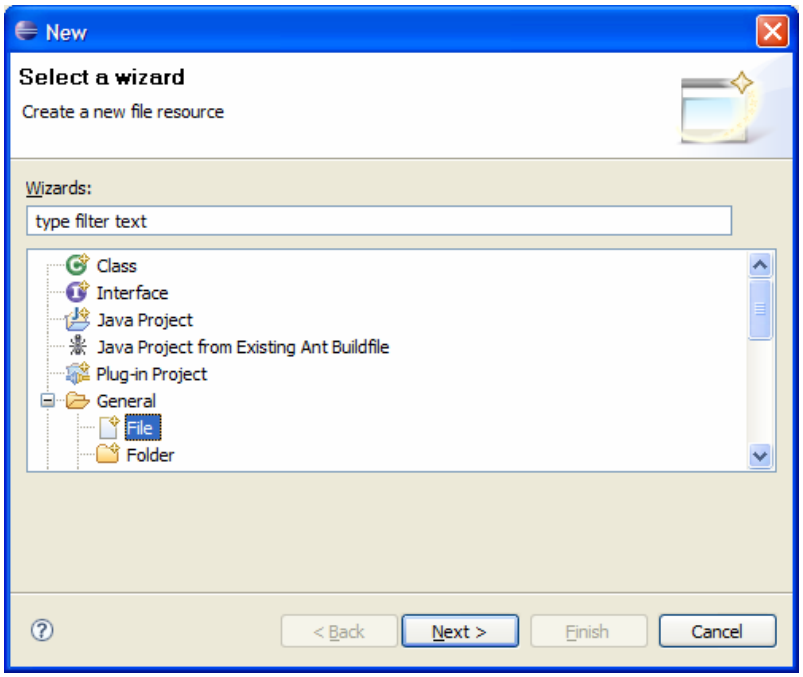


Figura 7.20: Menú de selección de asistente.

. . . o mediante el menú contextual «New» sobre un proyecto o carpeta.

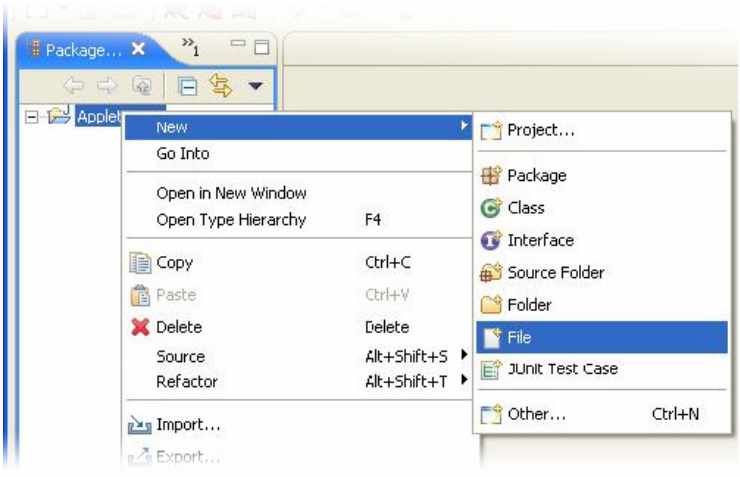


Figura 7.21: Menú contextual para la creación de un nuevo recurso.

En la ventana para la creación de un nuevo fichero se deberá indicar la carpeta donde

será guardado el fichero, así como el nombre y extensión de éste.

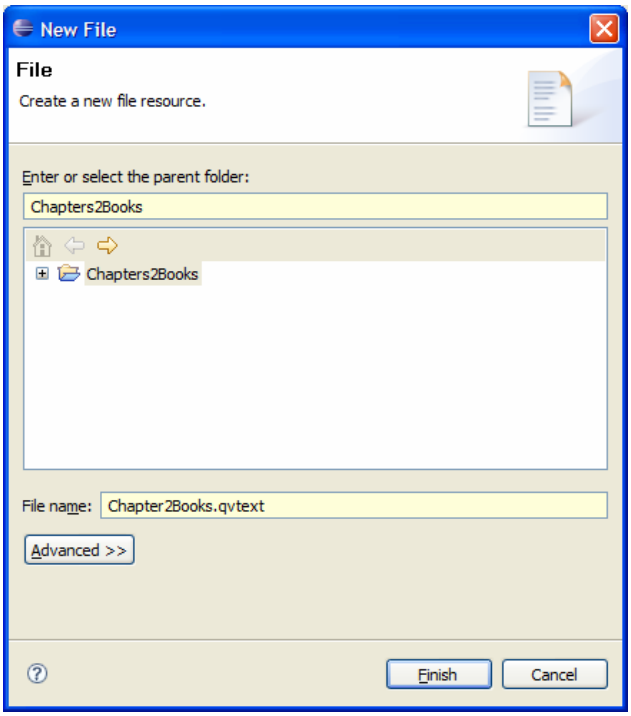


Figura 7.22: Ventana de creación de un nuevo archivo.

Cuando se cierra este cuadro de diálogo, el nuevo fichero se abrirá automáticamente en el editor de QVT. Éste es un sencillo editor textual con coloreado sintáctico para las palabras clave.

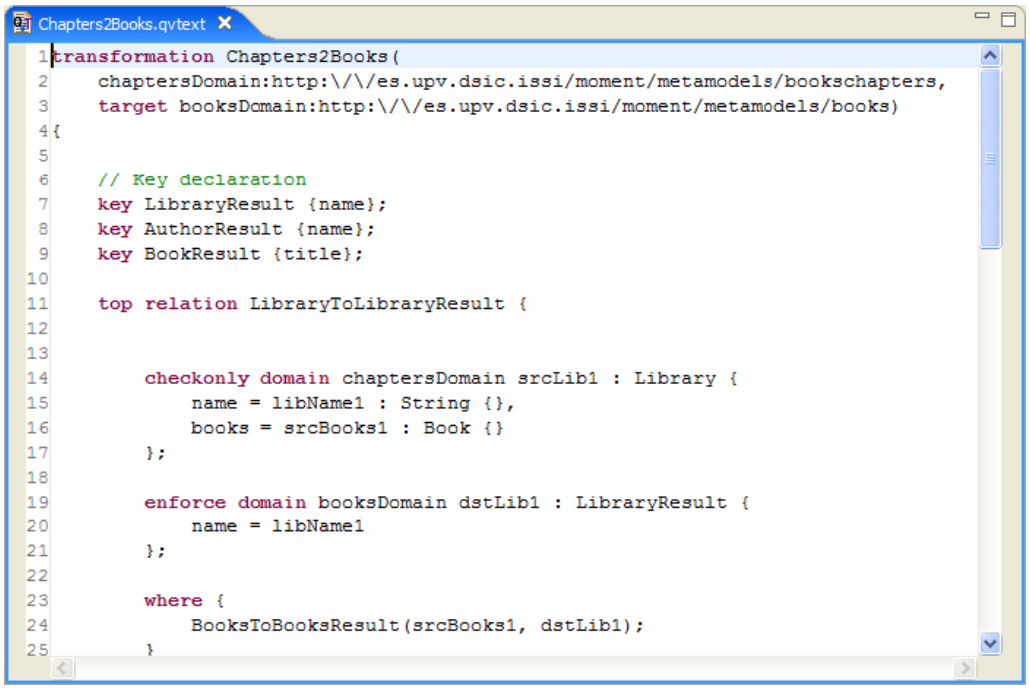


Figura 7.23: Vista del editor textual de QVT-Relations.

7.8.2. Obtención del código ejecutable final de una transformación.

Una transformación especificada en QVT-Relations es transformada en varios pasos hasta poder obtener el código que finalmente será ejecutado.

En MOMENT se ha definido un modelo ecore para representar el lenguaje QVT-Relations[6]. De esta forma, una transformación QVT se trata en EMF como instancia de este modelo. Esto permite trabajar con las transformaciones QVT a un mayor nivel de abstracción —en contraposición a otras representaciones, como un AST—, ya que disponemos de todos los mecanismos proporcionados por EMF para navegar la información sobre una transformación.

Por tanto, el primer paso de transformación consiste en analizar el programa QVT textual mediante un parser, y construir una instancia en EMF conforme al modelo «relations».

[6]La URI de este metamodelo es http:///es.upv.dsic.issi/moment/qvt/relations.

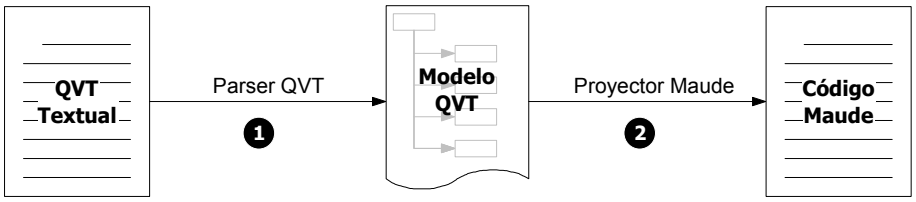


Figura 7.24: Pasos para la generación del código ejecutable final.

El modelo de la transformación QVT es el artefacto con el que MOMENT trata. En el momento de la ejecución (ver apartado 7.9, «Invocación de Transformaciones.»), es el fichero que determina la configuración del operador genérico ModelGen, y a partir del cual se obtiene el código Maude final.

> Es importante resaltar que en el paso de proyección a código Maude, no se realiza ninguna validación sobre la corrección o no del modelo QVT de entrada.

7.8.3. Proceso de creación del modelo.

Para crear un modelo a partir de la representación textual de una transformación QVT los metamodelos de los dominios implicados deben encontrarse registrados en EMF (ver apartado 7.4.4). En caso contrario, el proceso de análisis fallará.

El proceso de análisis de la transformación QVT textual se inicia mediante el menú contextual «QVT → Parse QVT textual program» (o la opción «QVT → Parse QVT textual program and show AST», que adicionalmente muestra el AST obtenido tras el paso de análisis sintáctico). Al invocarlo, los últimos cambios realizados a la transformación deben haber sido guardados en disco.

> Actualmente, el analizador de QVT sólo informa de errores sintácticos. Los errores semánticos (reglas que no se invocan, o se invocan con un nombre incorrecto, invocaciones con argumentos erróneos, variables no declaradas, etc.) serán pasados por alto, y no se informará de ellos en ningún paso posterior, causando que la transformación devuelva resultados incorrectos, o no devuelva ningún resultado en absoluto.

Los errores sintácticos detectados, se notificarán por la consola de MOMENT, en la vista de problemas de eclipse, así como mediante un marcador aproximadamente en la línea errónea en el margen izquierdo del editor.

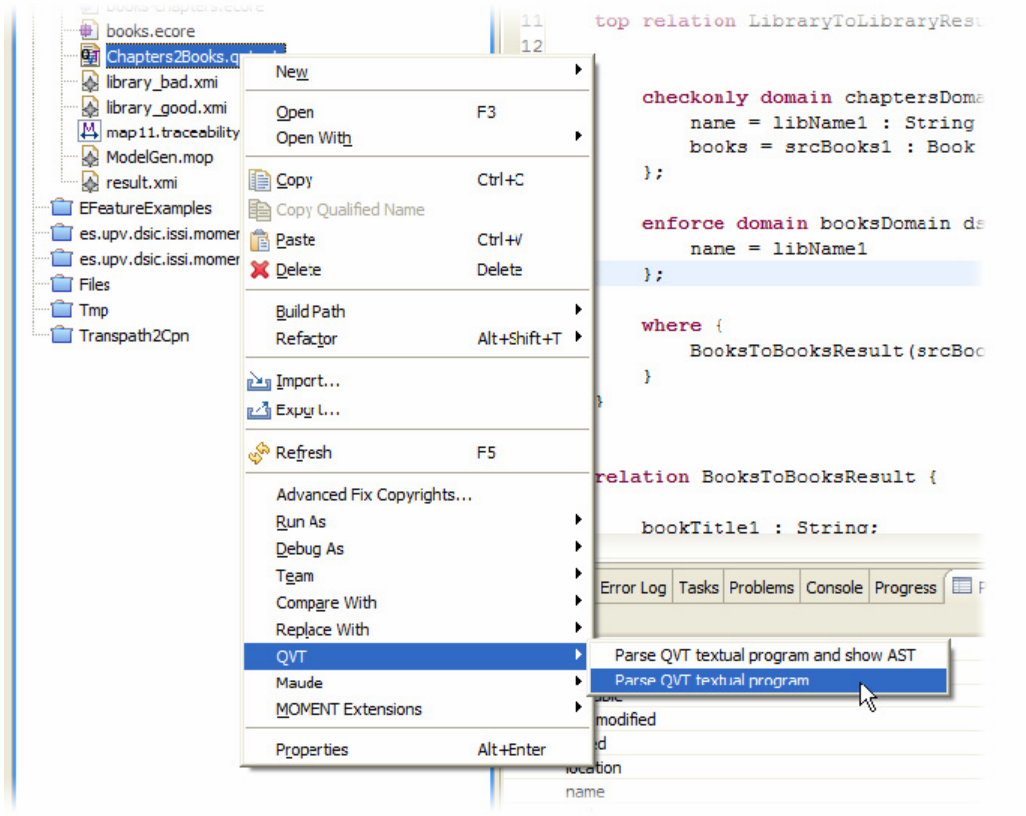


Figura 7.25: Menú contextual para analizar un programa QVT.

Los errores permanecerán notificados hasta que se realice de nuevo el proceso de análisis mediante el menú contextual.

7.9. Invocación de Transformaciones. Ejecución de la transformación Uml2Rdbms.

7.9.1. Creación del proyecto de ejemplo.

MOMENT proporciona un asistente que crea un proyecto con algunas transformaciones de ejemplo. Para iniciarlo deberemos abrir el asistente para creación de un nuevo elemento del workspace (CTRL+N o File → New → Other).

En la ventana de selección del asistente, marcaremos la opción «MOMENT → MOMENT QVT Examples» como muestra la figura 7.27:

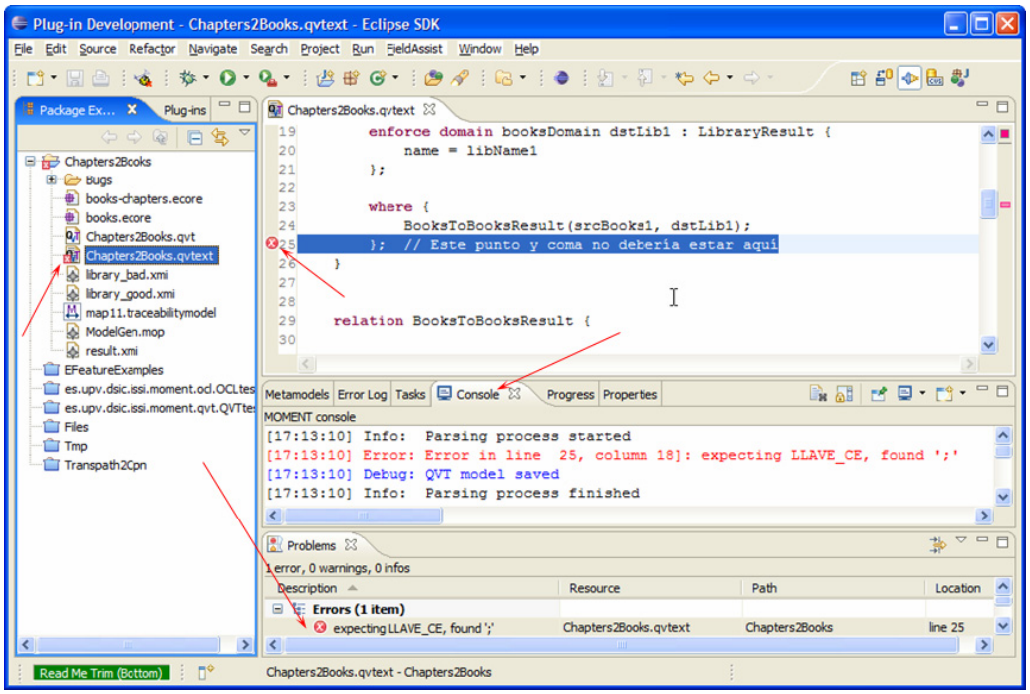


Figura 7.26: Error sintáctico en el editor de QVT.

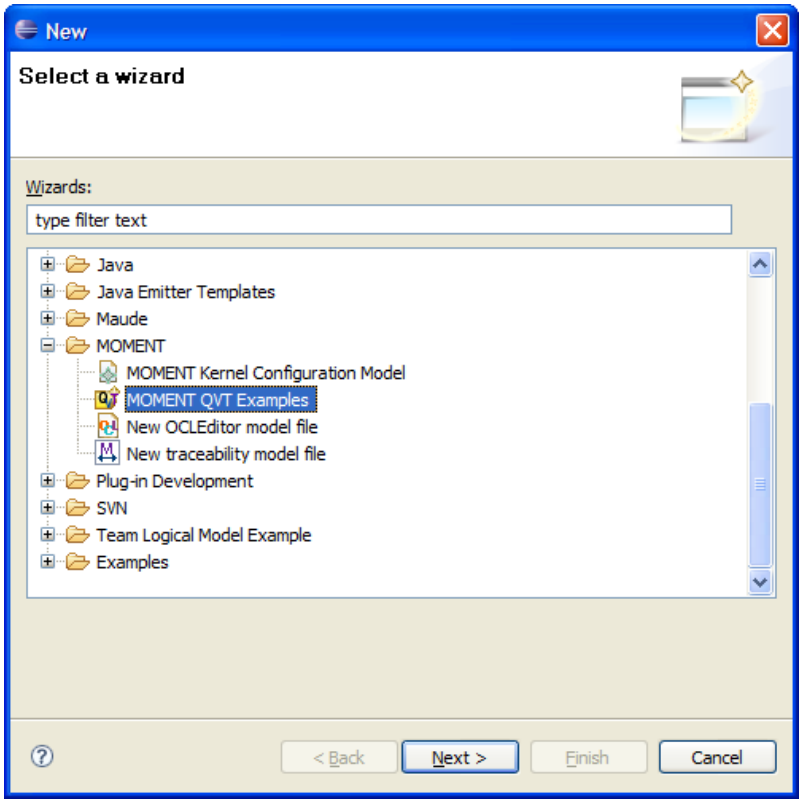


Figura 7.27: Selección del asistente de creación de un proyecto de ejemplos de QVT.

En la nueva ventana, indicaremos el nombre del proyecto que deseamos crear, y finalmente, pulsaremos «Finish».

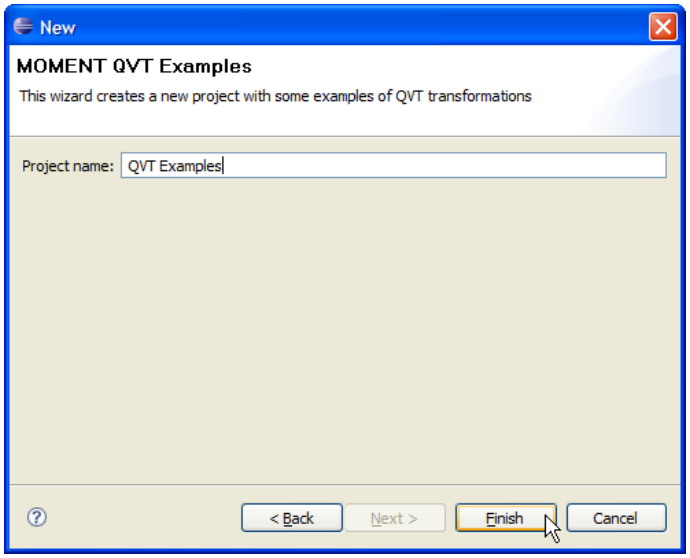


Figura 7.28: Asistente de Instalación.

En este punto, comenzará la creación del proyecto. En la ventana de navegación del workspace aparecerá el nuevo proyecto con los siguientes contenidos:

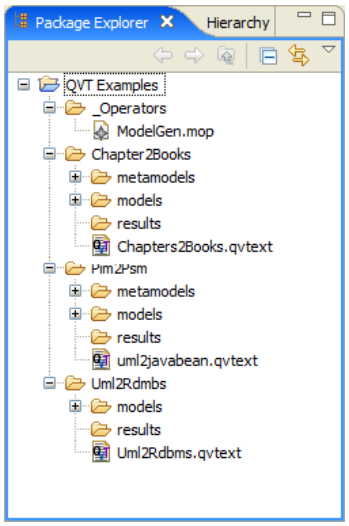


Figura 7.29: Ficheros del proyecto de ejemplo de transformaciones QVT.

7.9.2. Preparación para la ejecución de transformaciones.

La siguiente sección muestra cómo se invoca una transformación en MOMENT.

Como ejemplo se emplea la transformación «Uml2Rdbms» incluida en los ficheros de muestra proporcionados. La figura 7.30 muestra una captura del workspace mostrando el editor textual de QVT.

También se pueden observar abajo las diferentes vistas abiertas. En el uso de MOMENT es interesante tener disponibles las vistas:

1. *Problems*. Muestra los mensajes de error sintácticos de los programas QVT.
2. *Metamodels*. Muestra la lista de metamodelos disponibles en EMF.
3. *Console*. Muestra información acerca del proceso de ejecución.
4. *Progress*. Muestra el progreso de la ejecución de transformaciones.

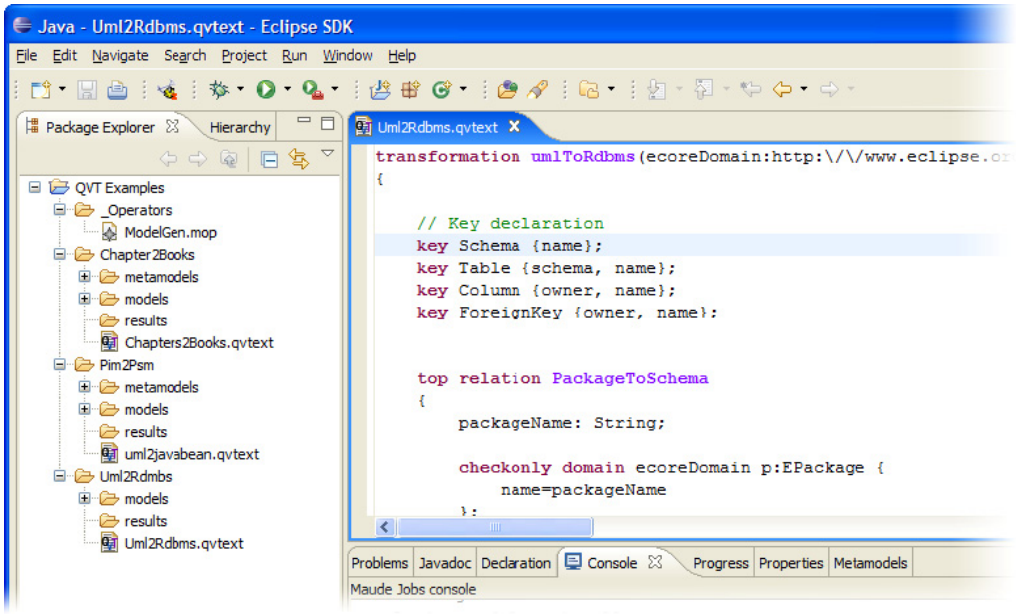


Figura 7.30: Vista general de un workspace con las transformaciones de ejemplo de MOMENT.

Para ejecutar la transformación primero se ha de obtener el modelo QVT asociado a ella. Para ello haremos clic derecho sobre el fichero «*.qvtext» de la transformación (figura 7.31). Téngase en cuenta que los metamodelos involucrados deben estar registrados en EMF.

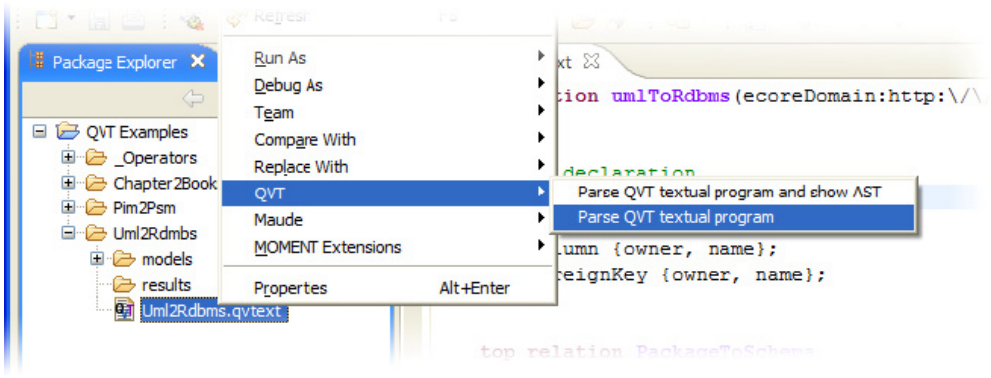


Figura 7.31: Parseado del fichero Uml2Rdbms.qvtext

Si el proceso de análisis ha concluido sin incidencias, se habrá creado un fichero en la misma carpeta, y con mismo nombre (pero extensión «*.qvt») conteniendo el modelo. Ha de recordarse que el proceso de análisis no hace comprobaciones de tipo semántico sobre la transformación.

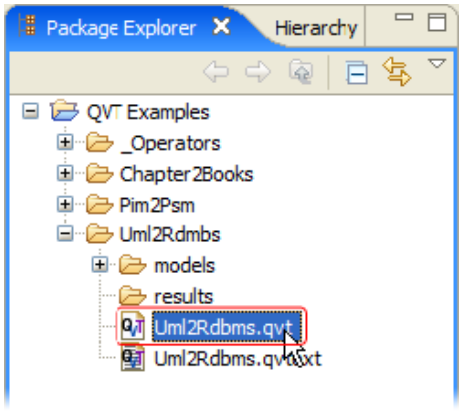


Figura 7.32: Fichero del modelo QVT generado para la transformación Uml2Rdbms.qvtext.

La interfaz de invocación de transformaciones de MOMENT está integrada en la interfaz estándar de eclipse para lanzar ejecuciones.

Para abrir el cuadro de diálogo correspondiente, podemos hacer uso del menú desplegable de la barra de herramientas como muestra la figura 7.33.

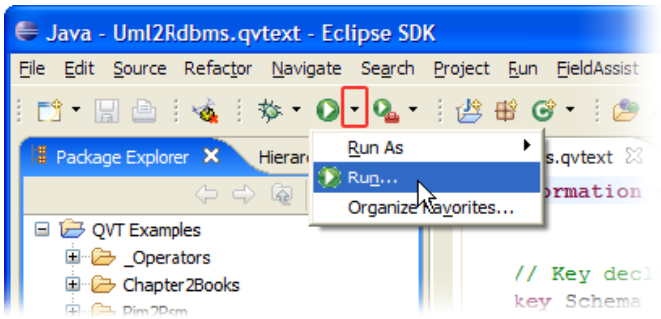


Figura 7.33: Apertura de la ventana de ejecución de Eclipse.

Alternativamente a este método, podemos lanzar la ejecución directamente haciendo clic derecho sobre el fichero del operador ModelGen.

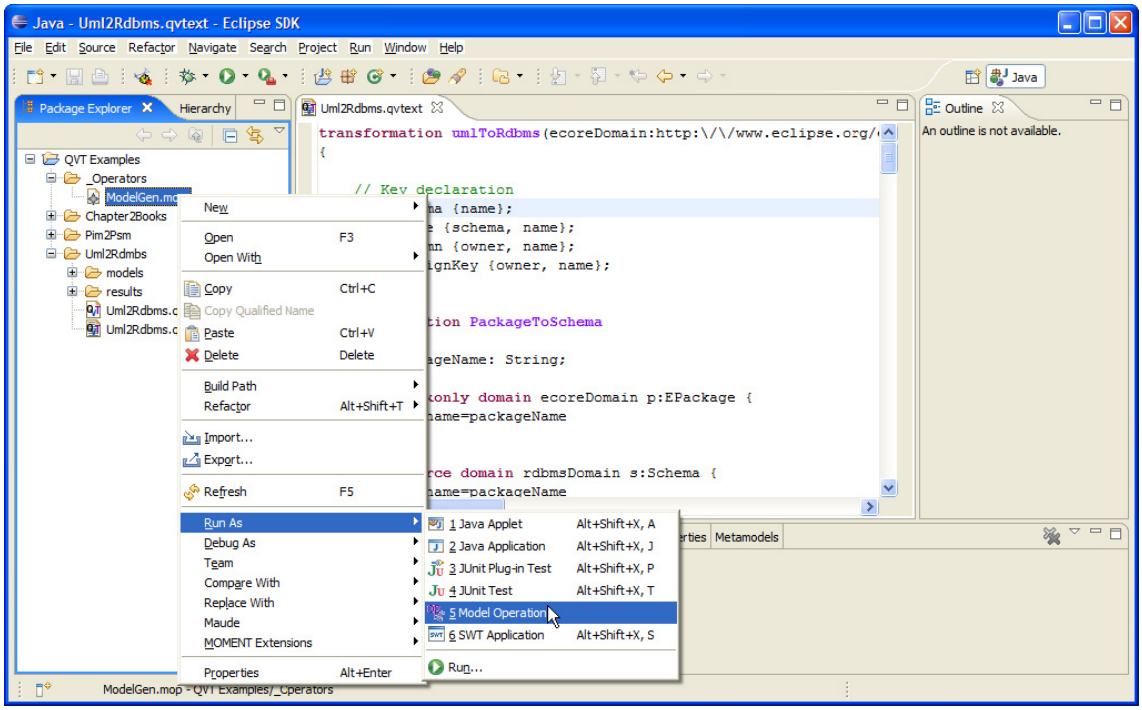


Figura 7.34: Creación de una nueva invocación a partir del operador ModelGen directamente.

7.9.3. Configurando la invocación de transformaciones.

Una vez abierta la interfaz de ejecución, se debe crear una nueva invocación de un operador. Para ello, se selecciona el ítem «Model Operation» y se hace clic sobre el botón «New launch configuration», como se muestra en la figura 7.35 izquierda. Tras pulsar el botón, se crea una nueva configuración como se muestra en la figura 7.35 derecha.

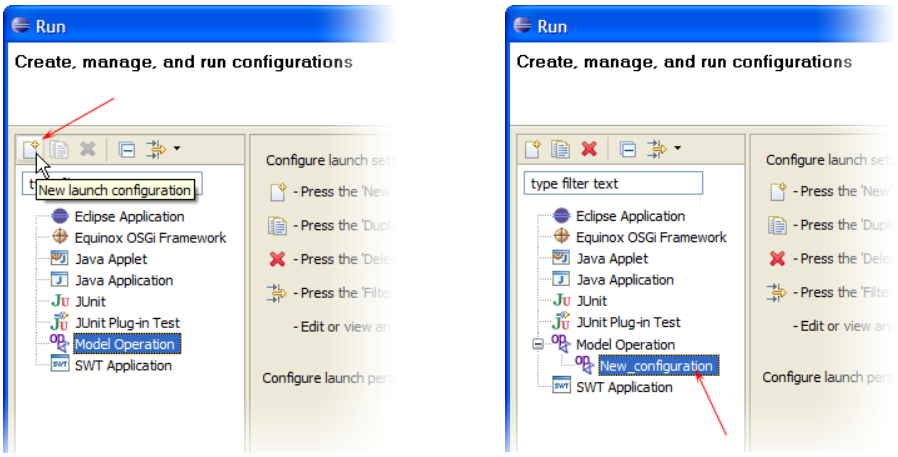


Figura 7.35: Creación de una nueva invocación.

Una vez se ha creado la nueva configuración para la ejecución de un operador, el cuadro de diálogo tendrá un aspecto similar al de la figura 7.36. En este punto, se recomienda establecer el nombre de esta configuración, por ejemplo «Uml2Rdbms».

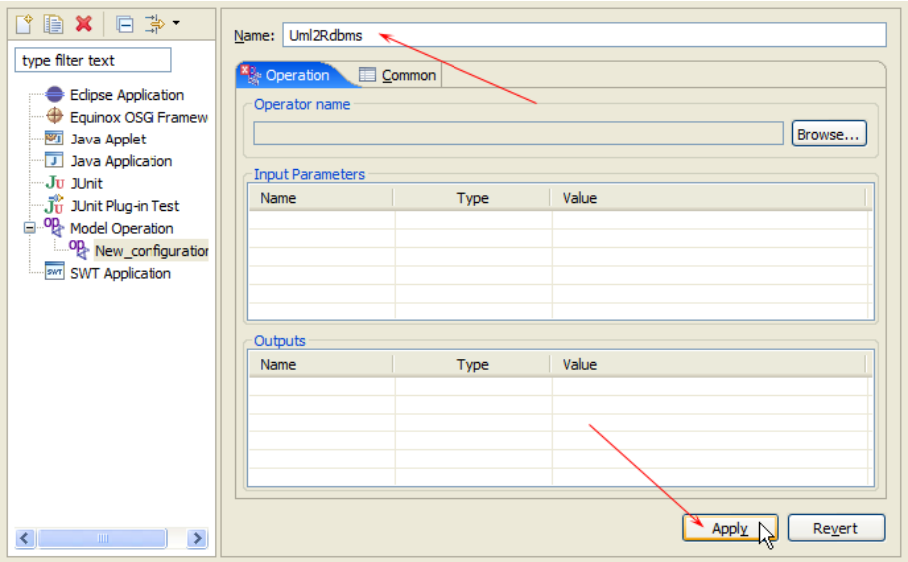


Figura 7.36: Establecimiento del nombre de la configuración de ejecución.

Para lanzar una transformación, deberemos cargar el operador genérico ModelGen. Para ello, pulsaremos el botón «Browse...»...

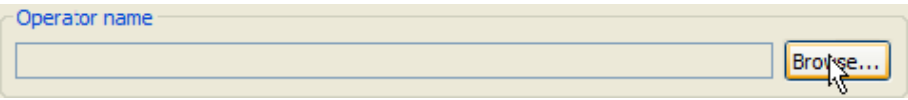


Figura 7.37: Selección del operador a ejecutar.

...y en el cuadro de diálogo de selección de fichero, marcaremos el fichero «ModelGen.mop» proporcionado entre los ficheros de ejemplo.

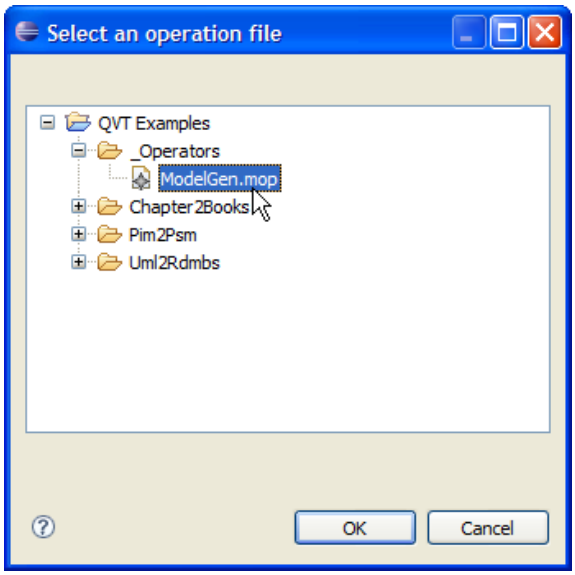


Figura 7.38: Selección del fichero del operador.

Una vez seleccionado el operador ModelGen, la interfaz de invocación modifica su aspecto como muestra la figura 7.39.

En este punto, puesto que el operador ModelGen es genérico, únicamente se puede determinar que devolverá un modelo resultado. Hasta que no se indique la transformación a realizar, no se podrá determinar ni el número ni el tipo de los argumentos de entrada.

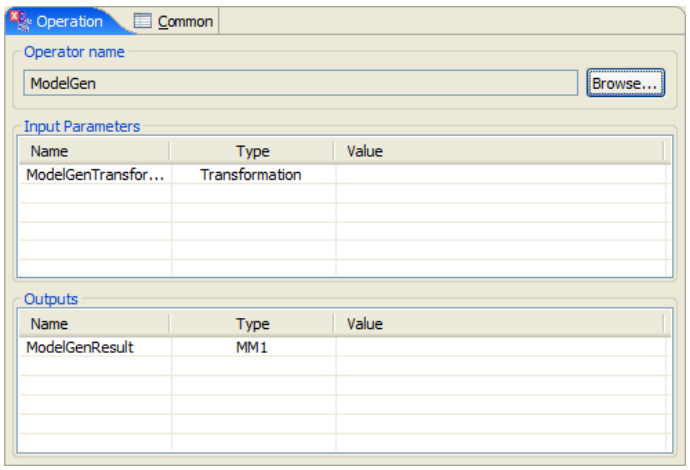


Figura 7.39: Vista de la ventana de ejecución del operador ModelGen inicialmente.

Para personalizar el operador ModelGen a una determinada transformación, deberemos seleccionar el fichero «*.qvt» que se ha obtenido anteriormente.

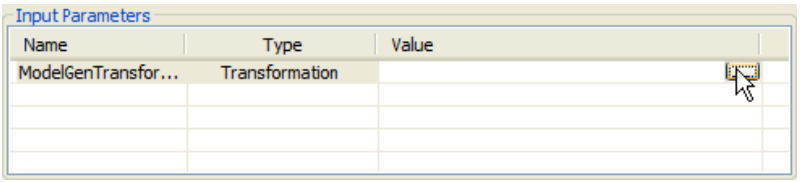


Figura 7.40: Configurando la ejecución del operador. Selección de la transformación.

Cuando se selecciona la casilla «Value» del campo «ModelGenTransformation», aparecerá el botón «...», que abrirá el cuadro de diálogo de selección de archivo (figura 7.41).

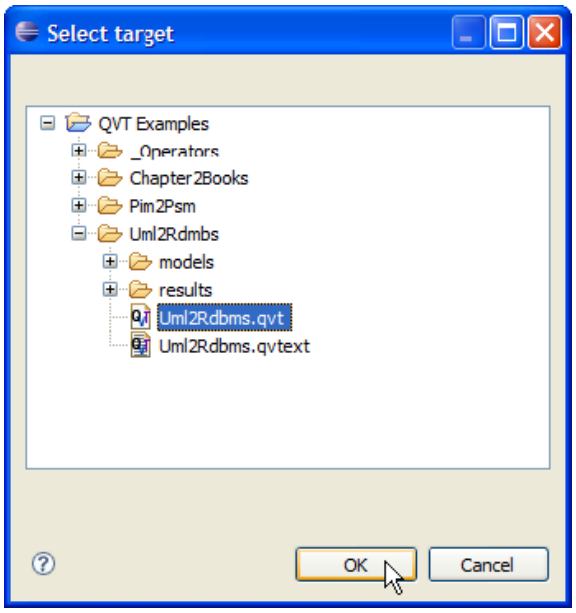


Figura 7.41: Selección del fichero de la transformación.

Al seleccionar una transformación QVT válida, el operador ModelGen se reconfigura, añadiendo a la lista de parámetros de entrada los dominios origen y destino de la transformación. En los parámetros de salida, además, se añade un campo para los modelos de trazabilidad. Se crean tantos modelos de trazabilidad como dominios origen se hayan definido en la transformación.

En caso de que se seleccione un fichero que no contenga una transformación, se mostrará un mensaje de error, y la ventana de configuración permanecerá sin cambios.

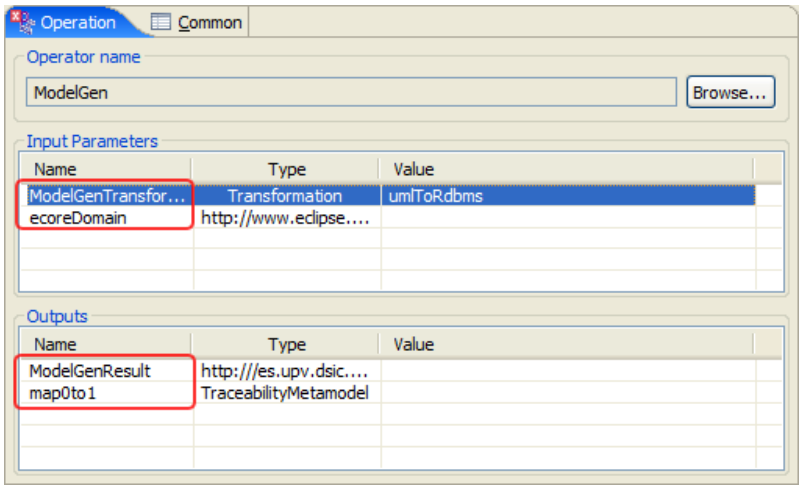


Figura 7.42: Vista del operador ModelGen personalizado para una transformación.

A continuación se han de completar los argumentos tanto de entrada como de salida. Para ello, se hace uso también del botón «...», como muestran las siguiente figuras.

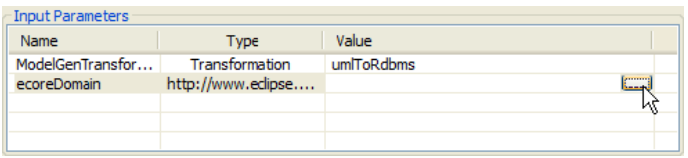


Figura 7.43: Establecimiento de los argumentos de entrada.

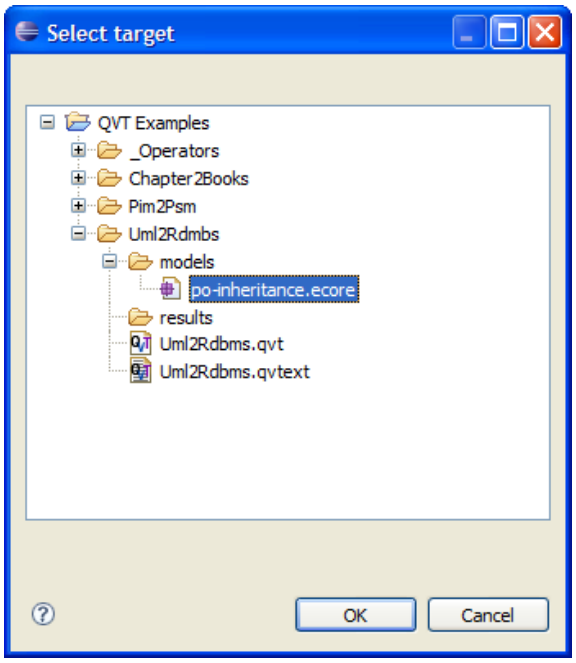


Figura 7.44: Selección del fichero del modelo origen.

Por último, se ha de indicar dónde se desea salvar los resultados de la transformación.

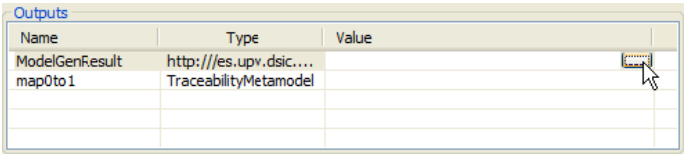


Figura 7.45: Establecimiento de los archivos de destino.

Haciendo uso del botón «  », se abre el diálogo de «Guardar como». En éste debemos indicar la carpeta dónde se va a salvar el fichero del modelo resultado, así como el nombre de éste en la casilla inferior. Si el modelo tiene un editor en árbol específico, se debe indicar la extensión del tipo de archivo al que se encuentra asociado dicho editor (por defecto, se corresponde con el valor nsPrefix especificado en el metamodelo). En caso de que se desee emplear el editor en árbol reflexivo de EMF, se puede indicar la extensión «*.xmi».

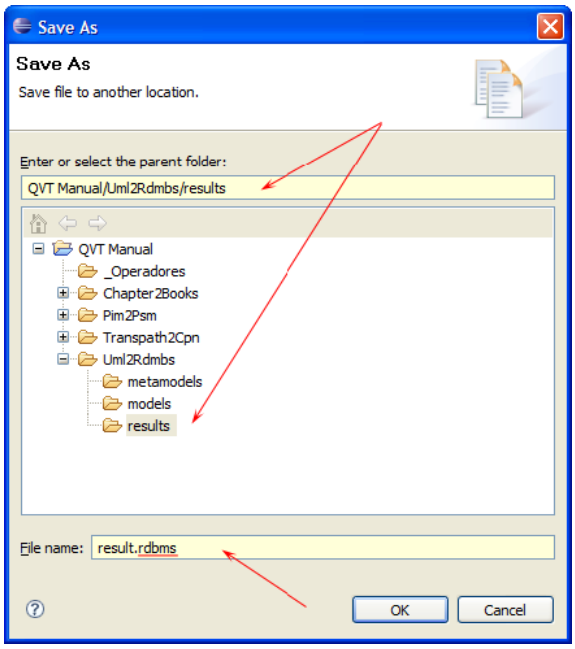


Figura 7.46: Diálogo de «Guardar archivo como...».

De la misma manera, se puede proceder a completar el resto de argumento de salida (modelos de trazabilidad). Los ficheros de los modelos de trazabilidad deben tener la extensión «*.traceabilitymodel» para que sean abiertos en el editor de trazabilidad de MOMENT.

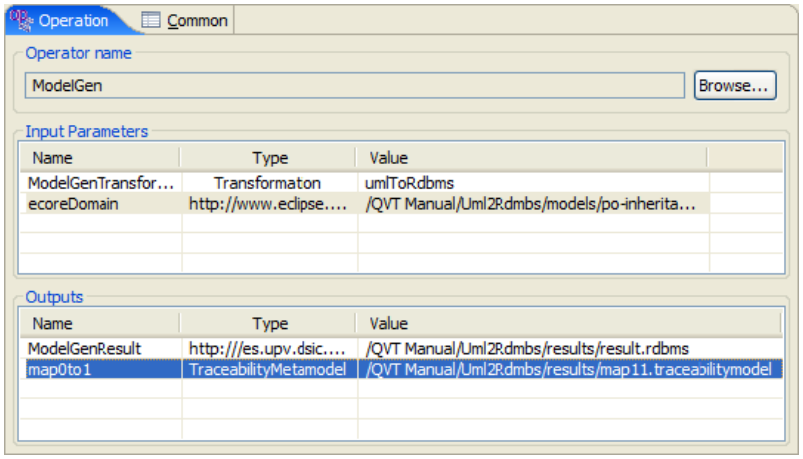


Figura 7.47: Operador ModelGen completamente configurado y listo para ejecutar.

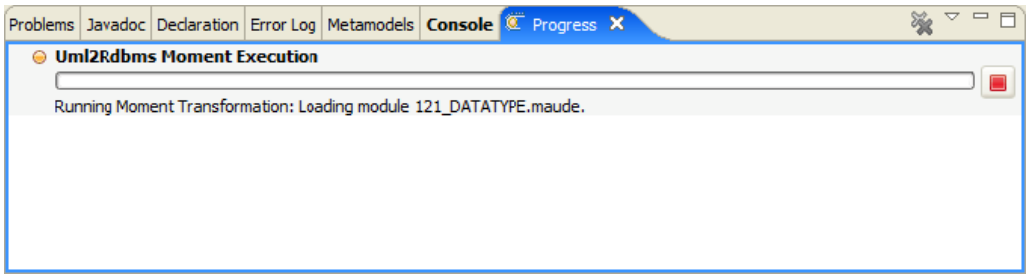


Figura 7.48: Vista de la barra de progreso.

Una vez se ha configurado por completo la invocación, se puede hacer clic en los botones «Apply» y «Run», de forma que se inicie la ejecución de la transformación.

Cuando se cambie una transformación, y se desee volver a ejecutarla, se deberá realizar estos pasos de nuevo, a pesar de que el fichero de la transformación sea el mismo. Los datos de una transformación están íntimamente ligados a la configuración de la invocación, puesto que pueden variar los dominios de ésta (nombre de los dominios, tipos, número, etc.).

Es por esto, que se ha decidido que la información de una transformación concreta forme parte de la invocación del operador ModelGen, siendo necesario recrear la configuración desde el comienzo.

7.9.4. Proceso de ejecución.

Cuando se ejecuta una transformación por primera vez, se inicia el kernel de MOMENT. Esto sólo ocurre una vez por sesión de trabajo. La barra de progreso estará activa durante todo el proceso de ejecución de la transformación, y muestra el proceso de carga del kernel, así como otra información. El valor del porcentaje de trabajo completado es orientativo.

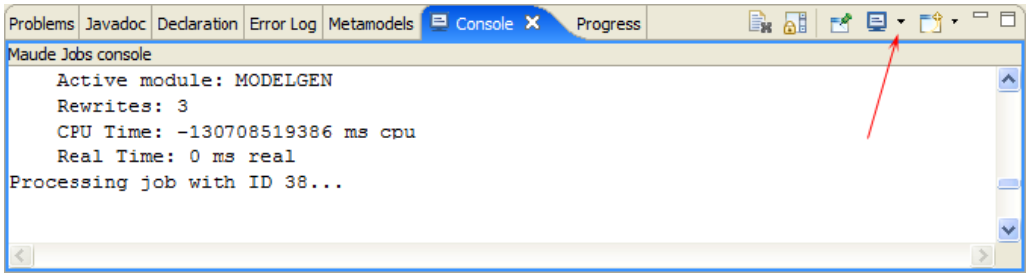


Figura 7.49: Vista de la consola de Eclipse. La consola de trabajos de Maude.

A parte de la barra de progreso se dispone de las consolas de Eclipse para notificar información al usuario. Eclipse comparte una única vista para todas las posibles consolas en funcionamiento, no obstante, sólo mostrará una de ellas. Mediante el botón resaltado en la figura 7.49 podemos desplegar el menú de selección de consola activa, como muestra la figura 7.50.

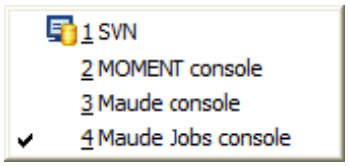


Figura 7.50: Selección de la consola activa.

MOMENT dispone de tres consolas donde se muestra distinta información. Éstas son:

1. Consola de MOMENT. En ella se volcarán los mensajes de error así como otra información de progreso o de depuración. Éste es configurable. Mediante la ventana de preferencias («Window → Preferences . . . → MOMENT → Console») se pueden seleccionar tanto los tipos de mensajes que deseamos visualizar, como los componentes de MOMENT que se desea que muestren información.

2. Consola de Trabajos de Maude. Muestra información acerca de los trabajos enviados a Maude, esto es, carga de módulos, último módulo activo, etc. Cuando se envían comandos de reescritura o reducción, informa acerca del tiempo empleado y reescrituras realizadas.

3. Consola de Maude. Muestra los último comandos enviados a Maude y devueltos por el proceso sin ningún tipo de filtrado. Es últil para determinar cuándo una transformación ha fallado mediante la inspección del código Maude directamente. En caso de que la información que muestre esta consola sea insuficiente, se puede consultar el fichero de registro de ejecución de Maude que se haya indicado en las preferencias.

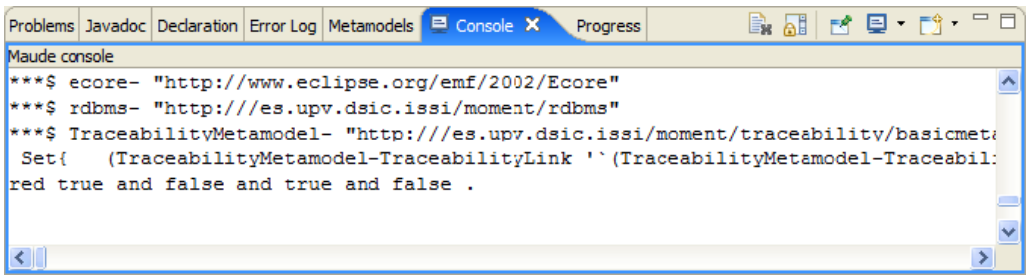


Figura 7.51: Vista de la consola de Maude.

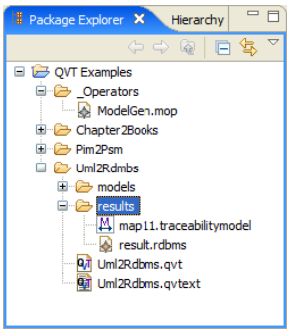


Figura 7.52: Explorador de archivos. Ficheros resultantes de la transformación.

7.9.5. Fin de la ejecución.

Cuando la ejecución de la transformación ha terminado (la barra de progreso desaparece), los ficheros de los modelos resultado aparecen en la vista del explorador de Eclipse.

La figura 7.52, muestra el fichero de resultado obtenido tras la ejecución de la transformación de ejemplo.

Si se hace doble clic sobre el modelo de trazabilidad (y éste tenía la extensión correctamente establecida), se abrirá el editor de trazabilidad, mostrándonos las correspondencias entre los elementos del modelo origen y el modelo destino. El editor de trazabilidad permite navegar estos enlaces.

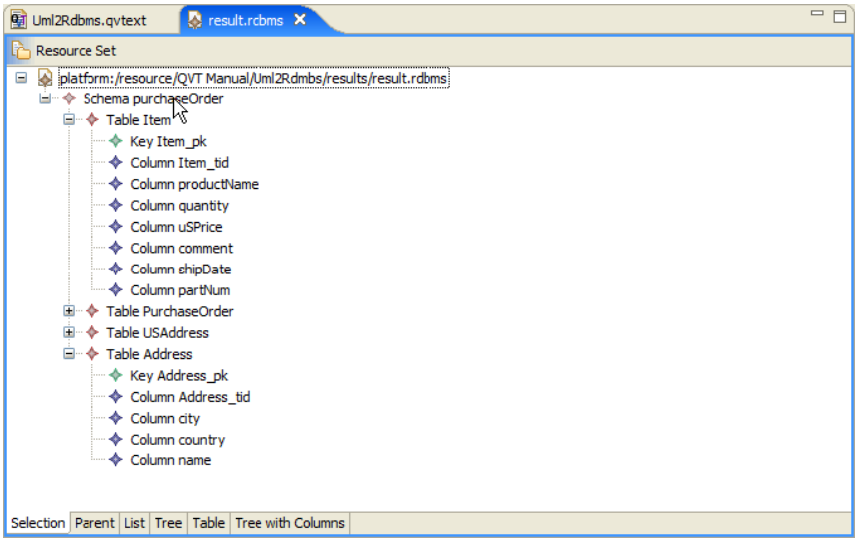


Figura 7.53: Vista del modelo resultante.

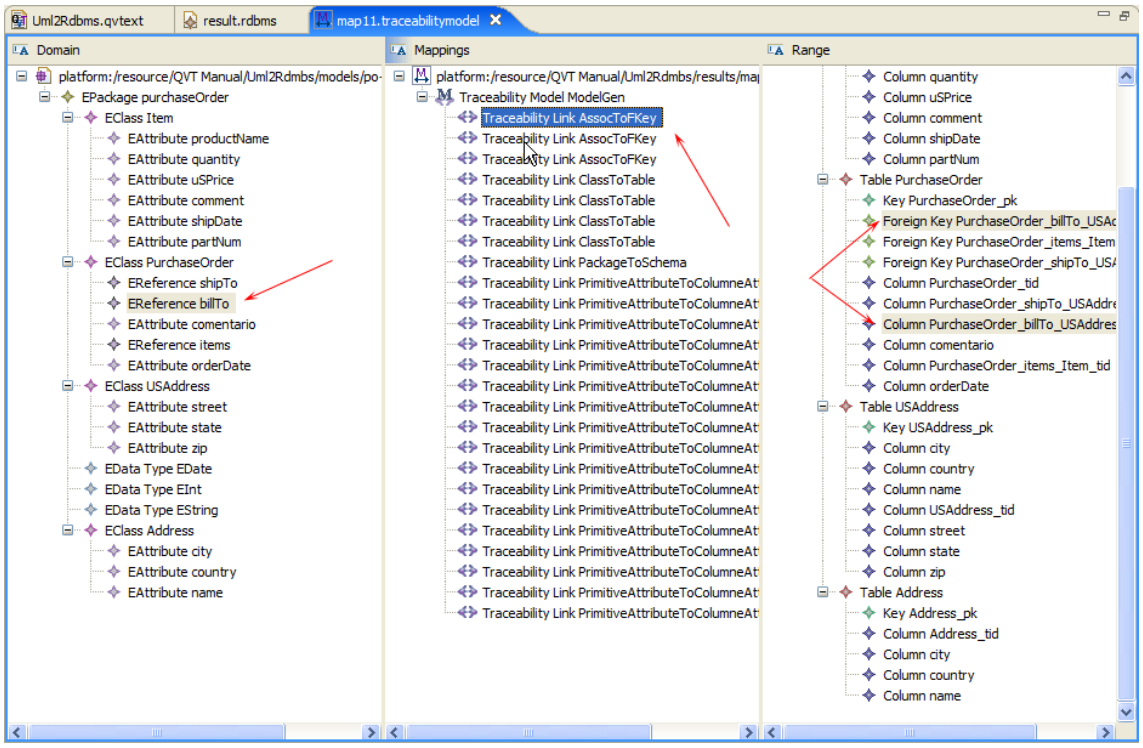


Figura 7.54: Vista del modelo de trazabilidad resultado.

Como se muestra en la figura 7.54, al hacer clic en un determinado link, se resaltan tanto en el modelo dominio como en el modelo rango cuáles son los elementos que se relacionan mediante dicha regla.

Si se hace clic en un elemento del modelo dominio o en un elemento del modelo rango, también se resaltarán los elementos que estén relacionados con el elemento seleccionado.

Capítulo 8

Manual del prototipo y ejemplo de ejecución usando Medini QVT

8.1. Obtención de la herramienta de ejemplo

La herramienta está disponible para su descarga en `ftp://lujuria.dsic.upv.es/transpath2cpn`. El archivo Comprimido `eclipse-platform-3.3.2-win32-Transpath2Cpn--Minimal.zip` contiene todos los archivos necesarios para ejecutae la transformación. A parte de ellos, incluye algunos archivos adicionales de ejemplo.

8.2. El *workspace*

La herramienta proporciona dos archivos de ejemplo:

- `example.xml`. Es un archivo XML extraído de la base de datos de TRANSPATH®. Contiene información acerca de un único pathway.
- `transpath2cpn.qvt`. Contiene la transformación QVT para transformar del dominio de TRANSPATH®al dominio de CPN Tools.

La figura 8.1 muestra el *workspace* de ejemplo que aparece cuando la herramienta se eejcuta por primera vez. El archivo `example.xml` está abierto con su editor por defecto (el editor de modelos de transpath). Este editor es capaz de representar la información de la base de datos de TRANSPATH®como instancias del metamodelo transpath. La figura 8.2 muestra el contenido *real* del archivo `example.xml`. como se puede observar, `example.xml` es un archivo XML extraído directamente de la base de datos TRANSPATH®.

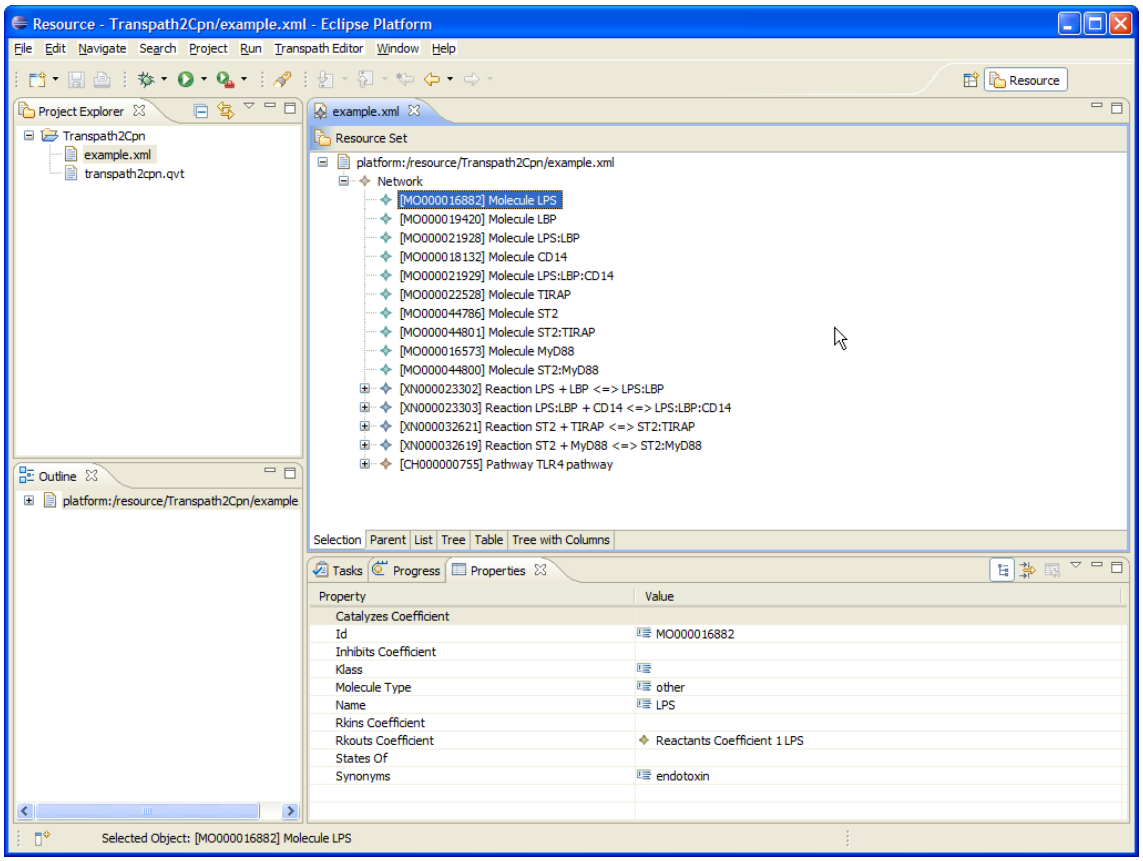


Figura 8.1: *Workspace* con los archivos de ejemplo.

Figura 8.2: Contenido real del archivo `example.xml`.

8.3. Ejecución del ejemplo

Para poder ejecutar la transformación, el usuario debe seleccionar Run as → 1 QVT Transformation tal y como se muestra en la figura 8.3.

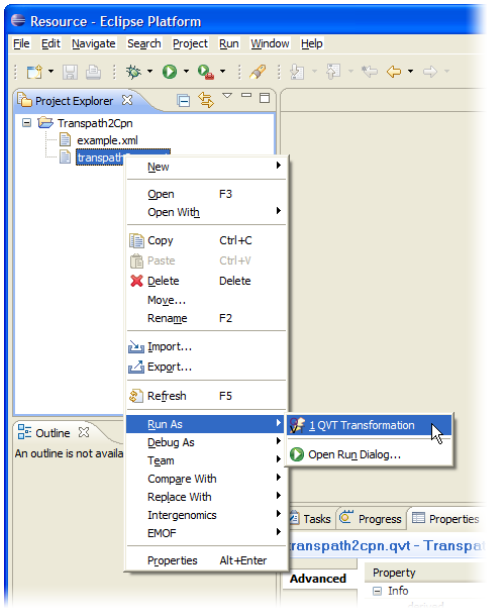


Figura 8.3: Ejecutando la transformación QVT.

Por su parte, la figura 8.4 muestra el cuadro de diálogo empleado para configurar las transformaciones. El campo editable que se encuentra en su parte superior contiene el nombre que identifica la transformación seleccionada en el panel izquierdo. Este nombre se inicializa al nombre del la tranformación seleccionada al abrir el cuadro de diálogo. El segundo campo, (que contiene el botón Browse...) a su derecha) indica la ruta del archivo que contiene la transformación a ejecutar. el botón Browse... se emplea para seleccionar el archivo de la tranformación a ejecutar (archivo `<<*.qvt>>`). Cuando se abre el dálogo Run as partiendo del menú contextual, este valor se establece automáticamente a la ruta del archivo seleccionado.

Una vez se ha seleccionado un archivo QVT, la tabla central puede emplearse para especificar qué archivos se corresponden con cada uno de los dominios de la transformación, tal y como la figura 8.4 muestra. Las figuras 8.5a y 8.5b muestran los cuadros de diálogo para especificar los dominios transpath y cpn respectivamente.

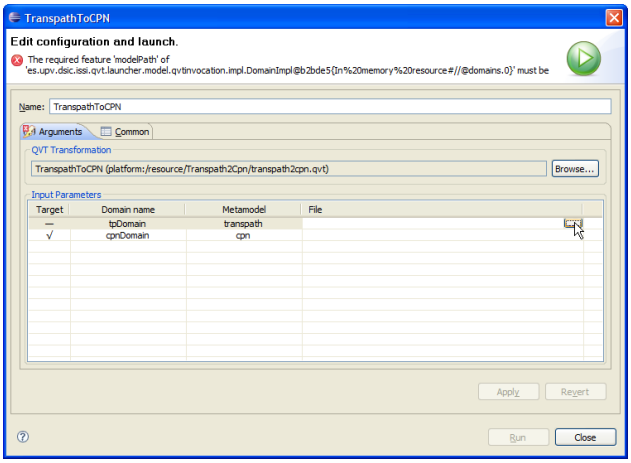
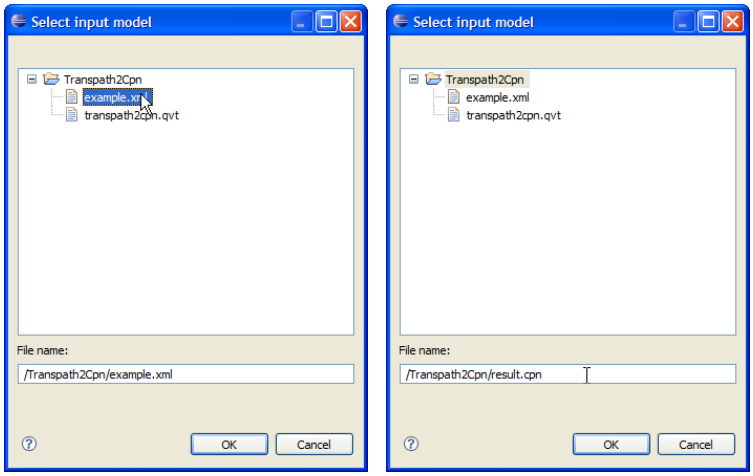


Figura 8.4: Configuración de la transformación.



(a) Archivo de entrada de la transformación (b) Archivo de salida de la transformación

Figura 8.5: Argumentos de la transformación.

La figura 8.6 muestra el cuadro de diálogo con toda la información necesaria para ejecutar la transformación. La dirección de la transformación viene determinada por la marca en la celda correspondiente de la columna *target* del dominio destino. Por defecto, el dominio destino es el último dominio definido en la transformación

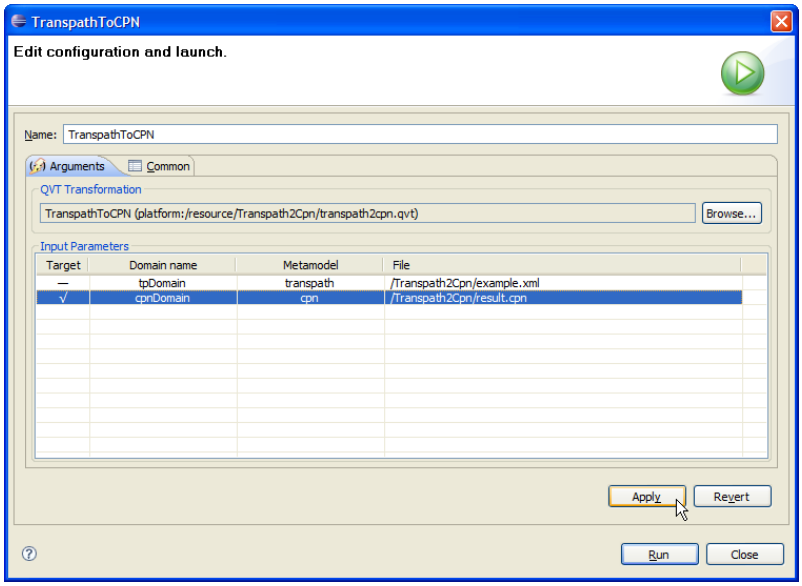


Figura 8.6: La transformación y sus argumentos.

8.4. Archivos de resultados

Una vez se ha configurado la transformación, se puede presionar el botón Run, iniciándose la ejecución de la transformación. cuando la transformación termina aparecen dos nuevos archivos en el *workspace*. El primero se corresponde con el modelo resultado (`result.cpn` para este ejemplo) y el segundo se corresponde con el modelo de trazas (`result.traces`, cuyo nombre se construye a partir del nombre del modelo). La figura 8.7 muestra la vista del explorador de proyectos con los archivos resultado.

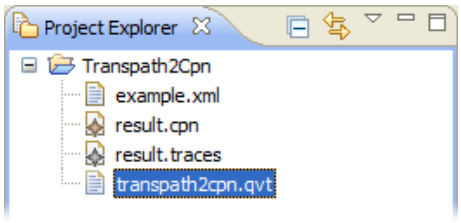


Figura 8.7: Archivos de resultado.

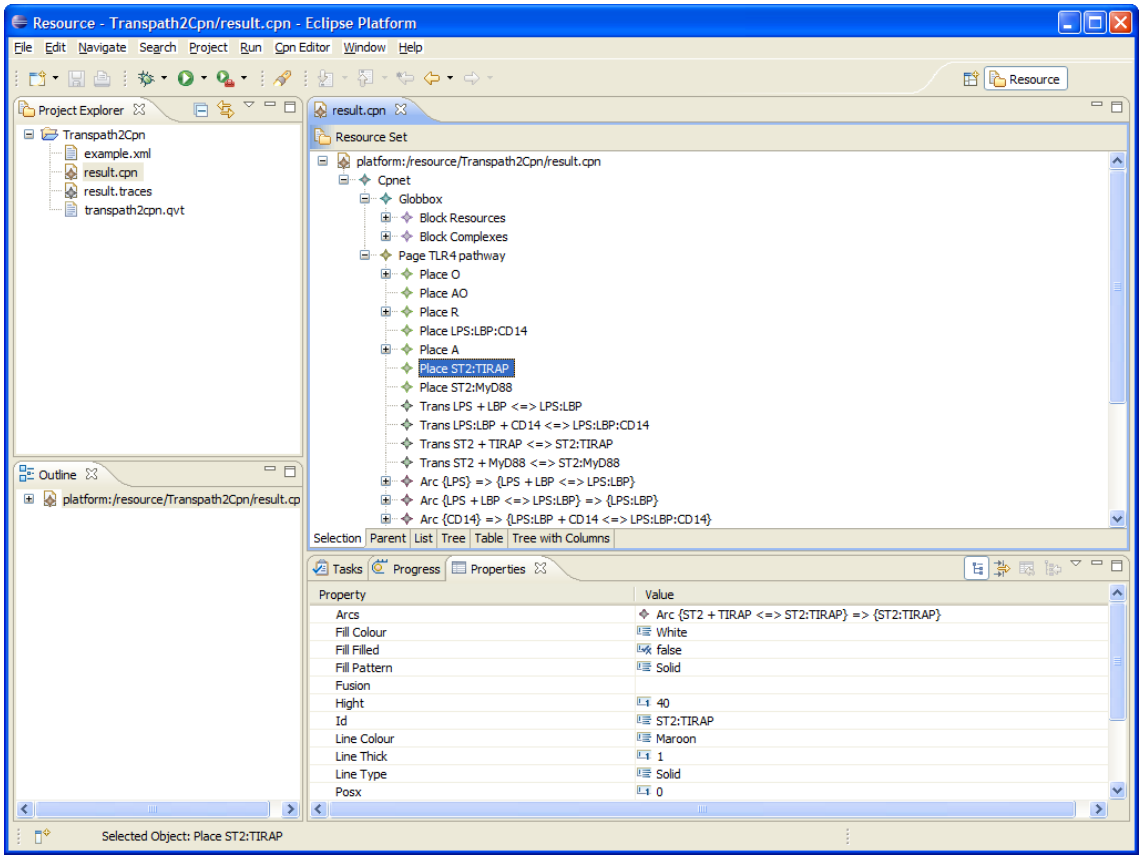


Figura 8.8: Editor de modelos de cpn.

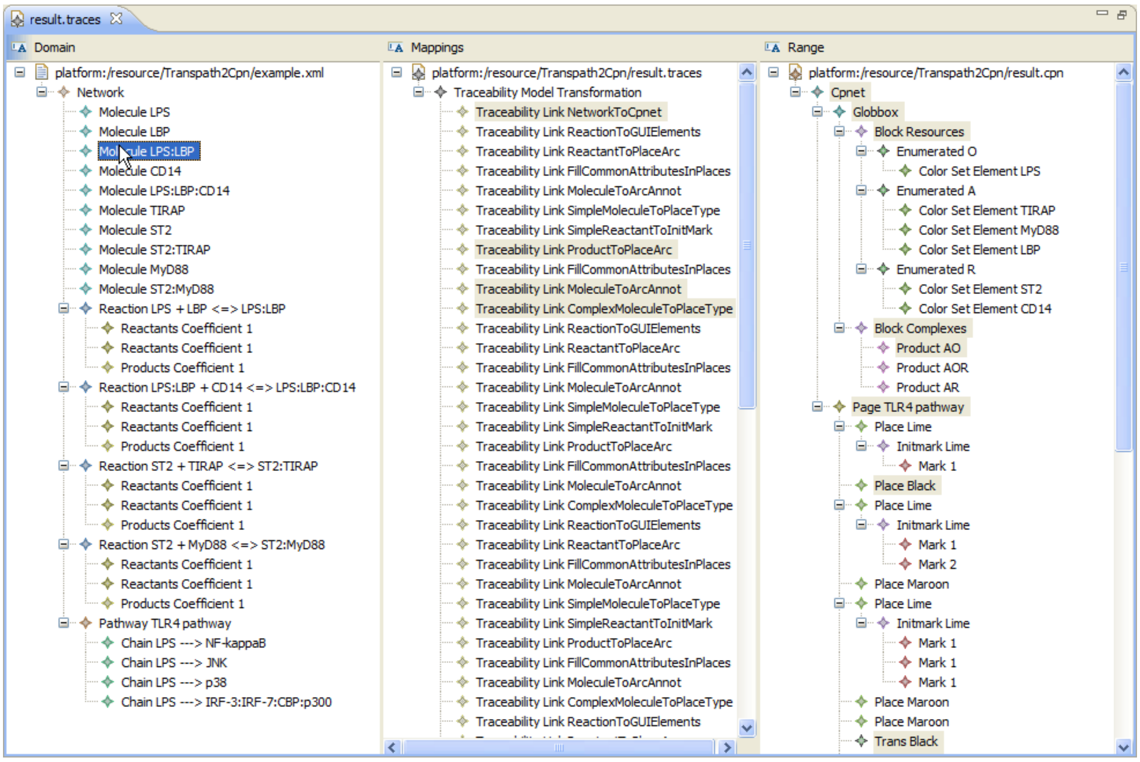


Figura 8.9: Editor de modelos de trazabilidad.

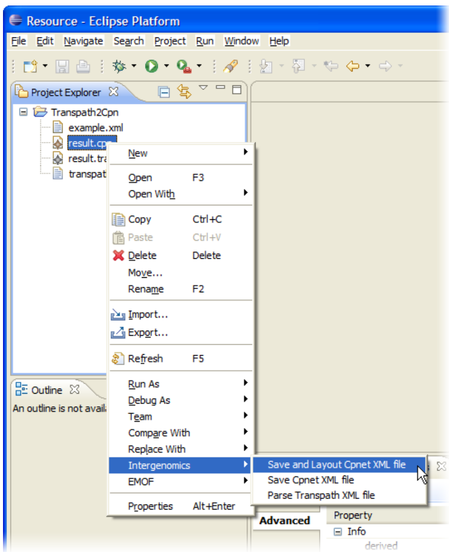


Figura 8.10: Exportar a CPN Tools.

El archivo de resultado en un documento XMO que es instancia del metamodelo de *CPN Tools*. La figura 8.8 muestra el archivo `result.cpn` en el editor de modelos (Cpn model editor), y la figura 8.9 muestra la modelo de trazas con el editor de trazabilidad por defecto. Este editor de trazabilidad muestra las correspondencias entre los dominios origen y destino.

Para poder abrir el modelo resultado en la herramienta *CPN Tools*, el archivo debe convertirse a un documento XML que sea válido para el esquema definido por dicha herramienta. Este paso se hace empleando el menú contextual (véase la figura 8.10). Además, en este punto también se puede aplicar un algoritmo de redibujado (*layout*) si el redibujado no se ha realizado durante el proceso previo de transformación.

En figura 8.11 se muestra el cuadro de diálogo para indicar el fichero XML final.

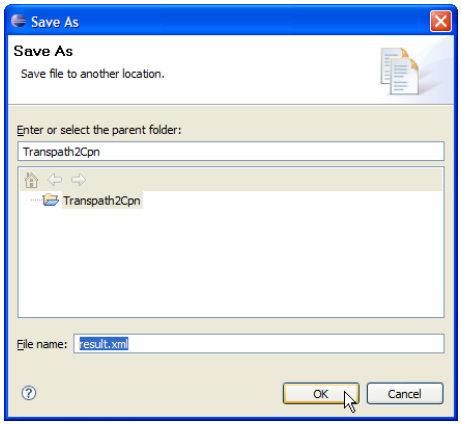


Figura 8.11: Guardar archivo como....

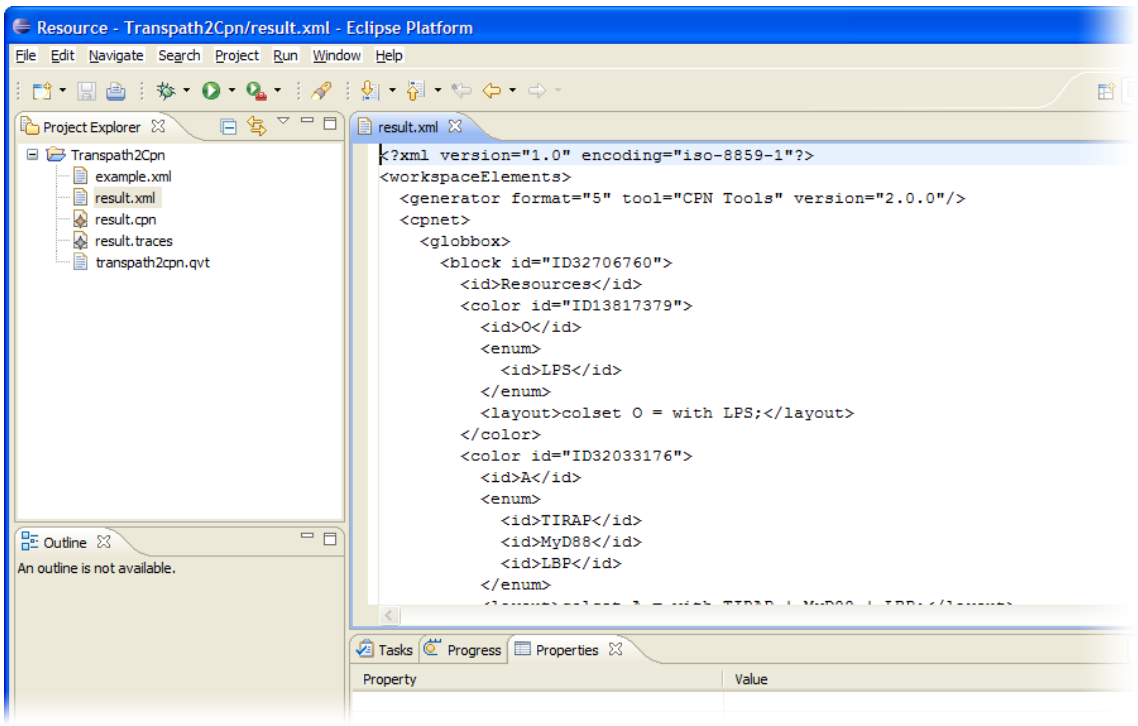


Figura 8.12: Contenidos del documento XML final.

La figura 8.12 muestra el documento XML final en el explorador de proyectos de Eclipse. Este archivo puede abrirse con el editor de texto por defecto, y, como se puede ver en la parte derecha de la imagen, contiene un documento XML válido que puede ser abierto directamente en *CPN Tools*.

8.5. Archivo resultado en *CPN Tools*

Finalmente, la figura 8.13 muestra cómo se representa la red de Petri. La posición de los lugares y la transiciones puede variar dependiendo de cómo se aplique el algoritmo de *layout*, ya que éste es no determinista. La red obtenida puede simularse directamente y sin necesidad de ninguna modificación en la herramienta *CPN Tools*.

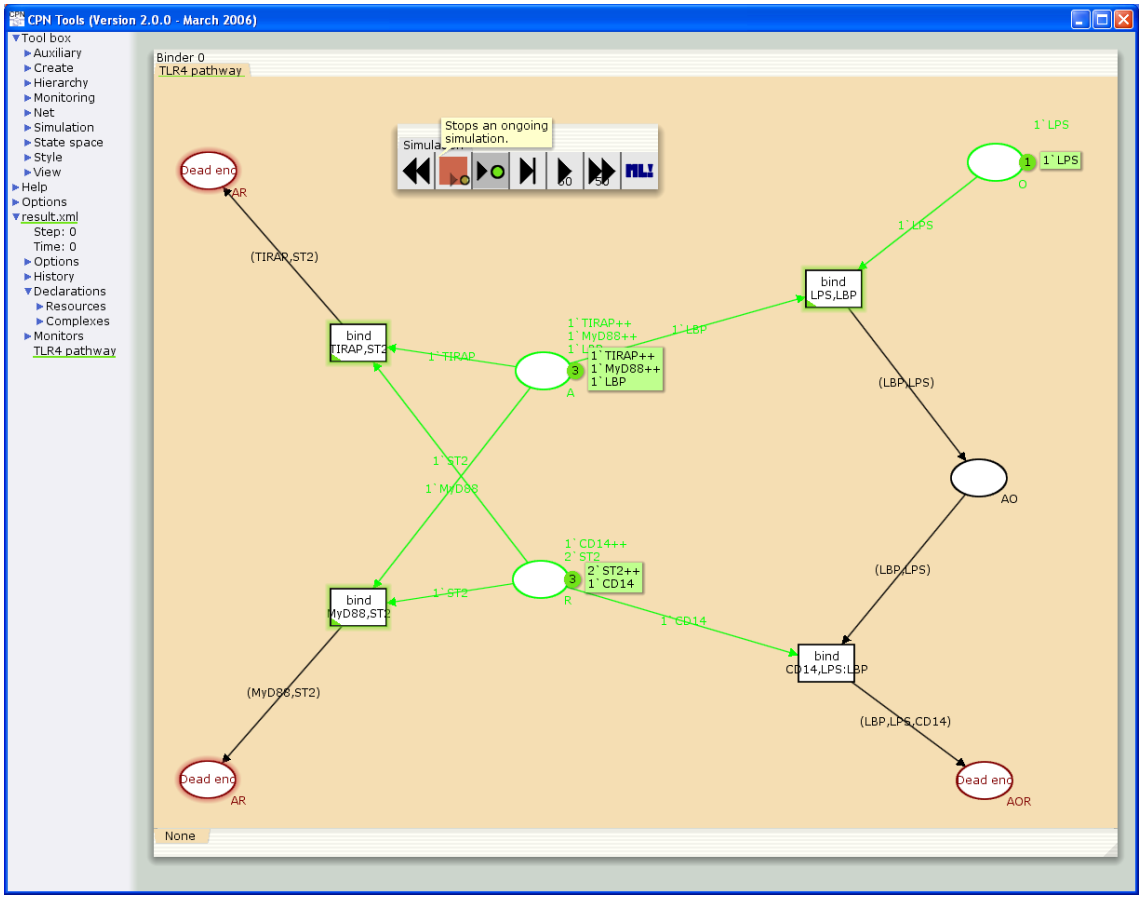


Figura 8.13: Resultado final en CPN Tools.

Capítulo 9

Conclusiones y trabajos futuros.

En este trabajo se ha presentado un caso de estudio en el cual se aborda un problema de interoperabilidad entre aplicaciones en el campo de la bioinformática mediante un enfoque dirigido por modelos. En este entorno es común la existencia de fuentes de datos y herramientas de simulación heterogéneas, y la natural representación de los datos biológicos como modelos permite abordar estos problemas desde un punto de vista más eficiente (acortando el ciclo de vida del desarrollo de software) y más elegante (puesto que las operaciones se realizan con un lenguaje más expresivo dada su naturaleza declarativa).

El trabajo pesenta como ventajas frente a las aproximaciones tradicionales (1) la automatización de tareas que anteriormente se realizaban de forma manual; (2) el desarrollo de herramientas modulares, independizando el mecanismo de transformación del formato de persistencia de los datos y facilitando la extensibilidad y mantenibilidad. (3) Aprovecha las ventajas de las transformaciones de modelos. El uso de modelos para representar los datos a ser transformados permite representar de forma más clara la estructura de éstos, siendo más intuitiva su manipulación ya que se trabaja con conceptos de alto nivel. (4) Se proporcionan capacidades de trazabilidad de forma implícita, ayudando a la localización de información errónea en los orígenes de datos. Por último, (5) el uso de un lenguaje como QVT-Relations ofrece como ventaja (frente a las aproximaciones imperativas tradicionales) que permite expresar de forma declarativa —y por tanto más descriptiva— las correspondencias entre los dominios origen y destino.

Respecto a la implementación realizada mediante MOMENT–QVT, ésta nos ha permitido depurar ampliamente la herramienta mediante un caso de estudio no trivial, y evitando emplear el clásico problema de transformación de UML a un esquema de bases de datos relacional. Además, el emplear una implementación tan pronta (realizada a mediados de 2006, sólo pocos meses después de la aparición del estándar final) nos ha permitido obtener un amplio conocimiento del lenguaje relations, facilitando el uso de otras implementaciones industriales más estables a pesar de las peculiaridades propias de cada implementación.

Respecto al lenguaje QVT-Relations, este ha mostrado diversas carencias en la apli-

cación al caso de estudio. En primer lugar, la implementación de los campos clave o *Keys* se ha mostrado problemática. Según el estándar, para toda clase que se emplee en el dominio destino debe de indicarse qué propiedad o conjunto de propiedades identifican unívocamente a cada objeto de dicha clase. Esto dificulta la creación de elementos duplicados (con los mismos datos), cosa que era necesaria en el caso de estudio. En cuanto a las implementaciones, en MOMENT-QVT se proporciona una implementación consistente con el estándar, esto es, nunca se crean dos elementos con el mismo valor en su campo clave. No obstante, el estándar es ambiguo en cuanto al tratamiento de estos casos, ya que no indica cómo proceder cuando se va a crear un elemento ya existente (reescritura, actualización sólo de los campos indefinidos, etc.). También se ha de tener en cuanta la estrategia al actualizar campos de multiplicidad múltiple (sobreescribir el valor, añadir la nueva referencia, etc.). En MOMENT–QVT se ha optado por añadir los nuevos valores a la colección de referencias.

En cuanto a MediniQVT, la implementación de los campos claves resulta poco consistente y propensa a comportamientos erráticos. En general, únicamente evita crear duplicados cuando los elementos que pueden ser potencialmente duplicados se crean en una misma regla. Esto es, que si en dos reglas distintas se crean dos objetos con los mismo valores en los campos clave estos se duplicarán, a pesar de la declaración de la *key*. Esto se puede evitar empleando condiciones en las cláusulas `where` donde se evalúe la condición *«campo».oclIsUndefined()* para comprobar que un elemento ya ha sido creado. Respecto al las referencias con multiplicidad mayor que uno, en MediniQVT se ha optado por sobreescribir el valor de la referencia. Una posible solución para evitar la sobreescritura es emplear una expresión OCL en la siguiente forma: `«campo» = «campo»->including(«elemento_nuevo»)`.

Igualmente el uso de una aproximación completamente declarativa supone que la realización de ciertas tareas —que mediante una aproximación imperativa serían más sencillas— resulta más complejo. Un ejemplo de ello es la implementación del algoritmo de *layout* que no puede realizarse directamente en la transformación mediante QVT–Relations. Otro posible caso serían ciertas operaciones con cadenas de texto, o la generación de valores al vuelo —no derivados de otros—, como números aleatorios o autoincrementados. Este último caso sería otra posible solución para solventar las limitaciones en el uso de *Keys*, tal y como se hace en el modelo relacional de datos. La implementación de cajas negras según el estándar de QVT es otra posible solución a estas limitaciones, pero su uso y definicón es poco clara en el estándar y no existe en general soporte para ellas en las herramientas disponibles. En MOMENT–QVT específicamente no se da soporte de forma sencilla para la definición de cajas negras (aunque es posible añadir a mano código Maude se se opta por una ejecución manual, no obstante, esta solución es inviable para personas que no estén involucradas en el desarrollo de la propia herramienta MOMENT–QVT). Respecto a MediniQVT no se proporciona soporte explícito para cajas negras, aunque es posible ejecutar código Java arbitrario mediante una pequeña solución alternativa. En este caso, la solución consiste en definir modificar el metamodelo de un dominio de la transformación, añadiendo un método en su modelo Ecore. Posteriormente se puede modificar el código Java generado para el modelo Ecore, añadiendo en el cuerpo de dicho método el código Java deseado. Finalmente, este método puede invocarse desde la transformación

QVT. Este es el mecanismo empleado para ejecutar el algoritmo de *layout* dentro de la transformación en la implementación realizada en MediniQVT. No obstante, en este caso no resulta una buena solución, puesto que el algoritmo de redibujado es no determinista y no proporciona buenos resultados. Por otra parte, al ser la transformación declarativa, no es posible controlar el flujo de control de ésta, de manera que no es posible saber a priori cuántas veces se invocará el proceso de redibujado, resultando éste poco eficiente.

Respecto a la comparación de ambas soluciones propuestas, ambas presentan sus ventajas e inconvenientes. La principal ventaja del uso de MOMENT–QVT es el haber podido comenzar el trabajo al poco tiempo de aparición del estándar QVT. Esto ha permitido poder hacer uso de tecnicas de transformación basadas en estándares y no otros lenguajes propietarios (como por ejemplo ATL). Además, ha permitido ahondar en los problemas intrínsecos de las tecnicas de transformaciones de modelos, el tratamiento de los campos claves, actualización de elementos, etc. El que esta herramienta está basada en lógica de reescritura ha permitido obtener este prototipo tan tempranamente, siendo relativamente sencillo realizar la correspondencia entre reglas de QVT–Relations y reglas de transformación en lógica de reescritura. Además, dado que la implementación está realizada en Maude, es relativamente sencillo extender el conjunto de operadores básicos que se proporcionan al usuario (por ejemplo, extensiones a OCL). Otro aspecto favorable de la implementación en MOMENT–QVT es su soporte nativo paa modelos de trazabilidad y las herramientas asociados a ellos. En este sentido, el editor de trazabilidad y el soporte para la navegación de modelos han resultado muy útiles. Por otra parte, el principal escollo en el uso de la herramienta ha resultado la escasa información que se entrega al usuario en caso de error, y la eficiencia en la ejecución de transformaciones (ya que es necesario cargar muchos módulos previos a la ejecución).

La implementación realizada en MediniQVT ha permitido emplear un entorno para la definición de transformaciones mucho más amigable. La herramienta proporciona un editor con coloreado de sintaxis un mensajes de error sensiblemente más descriptivos que los proporcionados por MOMENT–QVT (no obstante, la implementación de MediniQVT también tiene numerosos casos en los que puede fallar sin proporcionar excesivo *feedback* sobre la causa real del error). Además, Las herramientas que dan este tipo de información al usuario son las que permanecen con el código cerrado, por lo que no se corresponden exactamente con las que se incluyen en el prototipo final aquí presentado. Por otra parte, la velocidad en la ejecución de las transformaciones es sensiblemente mayor, puesto que no hace uso de un sistema externo como es Maude, y no requiere de puentes de interoperabilidad (es una herramienta puramente implementada en Java). No obstante, el soporte para trazabilidad que se proporciona es notablemente inferior, motivo por el que se han tenido que adaptar las ricas herramientas de MOMENT-QVT a al motor de MediniQVT. Igualmente, dada la compleja infraestructura del motor, es más difícil realizar modificaciones y extensiones a éste para dar soporte a nuevas funcionalidades.

Entre ambos motores, la integración del motor de MediniQVT ha resultado más sencilla, pudiendo realizar un prototipo completamente automatizado que permite realizar la migración de datos en pocos *clicks* y en un tiempo despreciable.

Comparando el prototipo con la implementación diseñada en el trabajo de Ziegler [76] la solución propuesta en el presente trabajo presenta las siguientes ventajas:

1. En primer lugar, la definición de los datos origen y destino se ha hecho de forma más eficiente, puesto que se ha definido un modelo Ecore, y se ha generado de forma automática el código que da soporte a dicho modelo de datos sin tener que codificar una sola línea de código.

2. Por otra parte, el uso de modelos permite representar de forma más clara y estructurada los datos de los dominios origen y destino.

3. En tercer lugar, el uso de una aproximación declarativa permite expresar de forma mucho más clara y explícita las correspondencias entre los dominos origen y destino en contraposición a la aproximación de [76] donde se sigue una aproximación puramente imperativa.

4. Como consecuencia del punto anterior, tenemos que la transformación es mucho más concisa y clara en la implememementación que usa QVT–Relations comparada con la implementación imperativa. En este sentido, las transformaciones aquí presentadas expresan cómo se debe realizar la transformación en 400 líneas (en la solución de MOMENT–QVT) y 407 líneas (en la solución en MediniQVT). Por su parte, la implementación Java requiere de 1810 líneas únicamente en la clase principal que realiza la transformación (a lo que cabría sumarle diversas clases auxiliares).

5. La solución de Ziegler carece además de cualquier soporte a trazabilidad en contraposición al trabajo que aquí se presenta basado en MDE. En este caso, la trazabilidad es un aspecto crucial en este tipo de herramientas, puesto que presumiblemente se encontrarán fallos en las redes de Petri una vez se haya ejecutado la simulación. Para el caso de estudio es fundamental la trazabilidad puesto que una vez encontrado un error en los datos tras ejecutar la simulación es necesario poder trazar el origen del fallo para poder arreglar la inconsistencia en la base de datos TRANSPATH®.

6. Por otra parte, en la solución aquí presentada se reutilizan diversas bibliotecas que proporcionan un resultado sensiblemente superior. Este es el caso del uso de la biblioteca Jung [69], que permite obtener redes de Petri perfectamente posicionadas. En el caso de los trabajos de Ziegler no se ha implementado ningún algoritmo de redibujado. En este caso, se opta por colocar los elementos de forma secuencial a espacios de 50 píxeles.

7. En cuanto a la eficiencia, la solución que emplea el motor de MOMENT-QVT es sensiblemente más lenta que la presentada por Ziegler. No obstante, la implementación basada en MediniQVT emplea un tiempo similar que ésta, presentando ambas los resultados de forma inmediata.

Futuras líneas de investigación de esta aproximación dirigida por modelos para el desarrollo de aplicaciones en bioinformática van dirigidas en dos direcciones. En primer

lugar, la representación de los datos biológicos como modelos permiten el aprovechamiento de los nuevos *frameworks* para la generación de metáforas visuales a partir de éstos, como es el caso de *GMF*[13] o *MS DSL Tools*. Por otra parte las investigaciones en ingeniería dirigida por modelos pueden proporcionar un rico *background* tecnológico no únicamente para la transformación de datos de un dominio a otro, sino para tareas como la integración de estos obtenidos desde orígenes heterogéneos.

Capítulo 10

Agradecimientos

Agradecemos al Prof. Dr. H.D. Ehrich y M. Ziegler del *Institut für Informationssyeteme* de la *Technische Universität Carolo-Wilhelmina zu Braunschweig*, su apoyo, colaboración y experiencia aportados en el desarrollo de este trabajo. Igualmente agradecemos su duro trabajo a Pascual Queralt como soporte en el uso de MOMENT-QVT.

Bibliografía

[1] Biobase biological databases. `http://www.biobase-international.com`.

[2] Pathway logic web site. `http://pl.csl.sri.com/`.

[3] KathrynV Anderson, Gerd Jürgens, and Christiane Nüsslein-Volhard. Establishment of dorsal-ventral polarity in the drosophila embryo: Genetic studies on the role of the toll gene product. *Cell*, 42(3):779–789, 10 1985.

[4] Phillip A. Bernstein, Alon Y. Halevy, and Rachel A. Pottinger. A vision for management of complex models. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):55–63, 2000.

[5] Artur Boronat, José A. Carsí, and Isidro Ramos. Automatic support for traceability in a generic model management framework. In Alan Hartman and David Kreische, editors, *Model Driven Architecture - Foundations and Applications, First European Conference, ECMDA-FA 2005, Nuremberg, Germany, November 7-10, 2005*, volume 3748 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2005.

[6] Artur Boronat, José A. Carsí, and Isidro Ramos. Algebraic specification of a model transformation engine. In Luciano Baresi and Reiko Heckel, editors, *FASE*, volume 3922 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2006.

[7] Artur Boronat, José Iborra, José Ángel Carsí, Isidro Ramos, and Abel Gómez. Del método formal a la aplicación industrial en gestión de modelos: Maude aplicado a eclipse modeling framework. *Revista IEEE América Latina*, September 2005.

[8] Luca Cardelli. Abstract machines of systems biology. 3737:145–168, 2005.

[9] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[10] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[11] Danos and Laneve. Formal molecular biology. *TCS: Theoretical Computer Science*, 325, 2004.

[12] X. Du, A. Poltorak, Y. Wei, and B. Beutler. Three novel mammalian toll-like receptors: gene structure, expression, and evolution.

[13] Eclipse Organization. The graphical modeling framework, 2006. `http://www.eclipse.org/gmf/`.

[14] Sol Efroni, David Harel, and Irun R. Cohen. Toward rigorous comprehension of biological complexity: Modeling, execution, and visualization of thymic t-cell maturation. *Genome Research*, 13(11), november 2003.

[15] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications I*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. 1985.

[16] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, José Meseguer, and M. Kemal Sönmez. Pathway logic: Symbolic analysis of biological signaling. In *Pacific Symposium on Biocomputing*, pages 400–412, 2002.

[17] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, and Carolyn L. Talcott. Pathway logic: Executable models of biological networks. *Electr. Notes Theor. Comput. Sci*, 71, 2002.

[18] EMF. `http://download.eclipse.org/tools/emf/scripts/home.php`.

[19] S. Rison S. Abiteboul et al. Emmett, S. Towards 2020 science. Technical report, Microsoft Corporation, 2006. `http://research.microsoft.com/towards2020science/downloads/T2020S_ReportA4.pdf`.

[20] CERN European Organization for Nuclear Research. The colt project. `http://acs.lbl.gov/~hoschek/colt/`.

[21] The Open Source Library for OCL Project. Oslo website. `http://oslo-project.berlios.de/`.

[22] Apache Foundation. Apache commons collections api. `http://commons.apache.org/collections/`.

[23] Apache Foundation. Apache logging services. `http://logging.apache.org/log4j/`.

[24] Apache Foundation. Xerces2 java parser. `http://xerces.apache.org/xerces2-j/`.

[25] Michael Y. Galperin. The Molecular Biology Database Collection: 2007 update. *Nucl. Acids Res.*, 35(suppl_1):D3–4, 2007.

[26] Abel Gómez. Proyecto final de carrera: Soporte gráfico para trazabilidad en una herramienta de gestión de modelos, 2005.

[27] Object Management Group. `http://www.omg.org`.

[28] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[29] C Hashimoto, S Gerttula, and KV Anderson. Plasma membrane localization of the Toll protein in the syncytial Drosophila embryo: importance of transmembrane signaling for dorsal-ventral pattern formation. *Development*, 111(4):1021–1028, 1991.

[30] Monika Heiner, Ina Koch, and Klaus Voss. Analysis and simulation of steady states in metabolic pathways with petri nets, August 02 2001.

[31] Scott E. Hudson. Cup parser generator for java. `http://www.cs.princeton.edu/~appel/modern/java/CUP/`.

[32] José Iborra. Proyecto final de carrera: Prototipo de integración de una herramienta de gestión de modelos, 2005.

[33] ikv++ technologies ag. ikv++ technologies ag website. `http://www.ikv.de`.

[34] Dragan Djurić J.M. Favreau Dragan Gašević Jean Bézivin, Vladan Devedžić and Frédéric Jouault. An m3-neutral infrastructure for bridging model engineering and ontology engineering. Geneva, Switzerland, feb 2005. Springer-Verlag.

[35] Na'aman Kam, Irun R. Cohen, and David Harel. The immune system as a reactive system: Modeling T cell activation with statecharts. In *HCC*, pages 15–22. IEEE Computer Society, 2001.

[36] Na'aman Kam, David Harel, Hillel Kugler, Rami Marelly, Amir Pnueli, E. Jane Albert Hubbard, and Michael J. Stern. Formal modeling of C. elegans development: A scenario-based approach. In Corrado Priami, editor, *CMSB*, volume 2602 of *Lecture Notes in Computer Science*, pages 4–20. Springer, 2003.

[37] Stuart Kent. Model driven engineering. In *Integrated Formal Methods, Third International Conference, IFM 2002, Turku, Finland, May 15-18, 2002, Proceedings*, volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer, 2002.

[38] Hiroaki Kitano. A graphical notation for biochemical networks. *BIOSILICO*, 1(5):169–176, 2003.

[39] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[40] Ivan Kurtev, Jean Bezivin, , and Mehmet Aksit. Technical spaces: An initial appraisal. *In Tenth International Conference on Cooperative Information Systems (CoopIS), Federated Conferences Industrial Track, California.*, 2002.

[41] Doug Lea. Biblioteca util.concurrent. `http://g.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html`.

[42] Ruslan Medzhitov, Paula Preston-Hurlburt, and Charles A. Janeway. A human homologue of the drosophila toll protein signals activation of adaptive immunity. *Nature*, 388(6640):394–397, 07/24 1997.

[43] Stephen J. Mellor, Scott Kendall, Axel Uhl, and Dirk Weise. *MDA Distilled*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[44] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 193–204, New York, NY, USA, 2003. ACM Press.

[45] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[46] F.Ñielson, H. R. Nielson, C. Priami, and D. Schuch da Rosa. Control flow analysis for bioambients. *Electronic Notes in Theoretical Computer Science*, 2003.

[47] Object Management Group. MDA Guide Version 1.0.1. 2003. `http://www.omg.org/docs/omg/03-06-01.pdf`.

[48] Object Management Group. UML 2.0 infrastructure specification (formal/05-07-05), 2004. `http://www.omg.org/cgi-bin/doc?formal/05-07-05`.

[49] Object Management Group. MOF 2.0 QVT final adopted specification (ptc/05-11-01). 2005. `http://www.omg.org/cgi-bin/doc?ptc/2005-11-01`.

[50] Object Management Group. Meta Object Facility (MOF) 2.0 Core Specification (ptc/06-01-01), 2006. `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`.

[51] Object Technology International, Inc. Eclipse platform technical overview, 2003. `http://www.eclipse.org/whitepapers/eclipse-overview.pdf`.

[52] Computing Laboratory University of Kent at Canterbury. Kent modeling framework. `http://www.cs.kent.ac.uk/projects/kmf/index.html`.

[53] OMG. *MOF QVT Final Adopted Specification*. Object Modeling Group, June 2005.

[54] M. Peleg, I. Yeh, and R. Altman. Modeling biological processes using workflow and petri net models, 2002.

[55] Priami, Regev, Shapiro, and Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *IPL: Information Processing Letters*, 80, 2001.

[56] The JDOM Project. Jdom website. `http://www.jdom.org/`.

[57] Pascual Queralt, Luis Hoyos, Artur Boronat, José Á. Carsí, and Isidro Ramos. Un motor de transformación de modelos con soporte para el lenguaje qvt relations. October 2006.

[58] Regev, Panina, Silverman, Cardelli, and Shapiro. Bioambients: An abstraction for biological compartments. *TCS: Theoretical Computer Science*, 325, 2004.

[59] Aviv Regev, William Silverman, and Ehud Y. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, pages 459–470, 2001.

[60] Mark Richters and Martin Gogolla. Validating UML models and OCL constraints. In Andy Evans, Stuart Kent, and Bran Selic, editors, *Proc. 3rd International Conference on the Unified Modeling Language (UML)*, volume 1939, pages 265–277. Springer-Verlag, 2000.

[61] Sun Microsystems, Inc. Javadoc java.util.regex.Pattern. `http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html`.

[62] Koichi Tabeta, Philippe Georgel, Edith Janssen, Xin Du, Kasper Hoebe, Karine Crozat, Suzanne Mudd, Louis Shamel, Sosathya Sovath, Jason Goode, Lena Alexopoulou, RichardA Flavell, and Bruce Beutler. Toll-like receptors 9 and 3 as essential components of innate immune defense against mouse cytomegalovirus infection. *Proceedings of the National Academy of Sciences*, 101(10):3516–3521, March 9 2004.

[63] Carolyn L. Talcott, Steven Eker, Merrill Knapp, Patrick Lincoln, and Keith Laderoute. Pathway logic modeling of protein functional domains in signal transduction. In *CSB*, pages 618–619. IEEE Computer Society, 2003.

[64] Claudia Taubner, Brigitte Mathiak, Andreas Kupfer, N. Fleischer, and Silke Eckstein. Modelling and simulation of the TLR4 pathway with coloured petri nets. *2006. EMBS '06. 28th Annual International Conference of the IEEE*, pages 2009–2012, August 2006.

[65] Claudia Taubner and Till Merker. Discrete modelling of the ethylene-pathway. In *ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops*, page 1152, Washington, DC, USA, 2005. IEEE Computer Society.

[66] The Eclipse Project. Asistente para la creación de un plugin EMF. `http://help.eclipse.org/help32/topic/org.eclipse.emf.doc/tutorials/clibmod/clibmod.html`.

[67] The Eclipse Project. Descarga de Eclipse 3.2.1. `http://download.eclipse.org/eclipse/downloads/drops/R-3.2.1-200609210945/`.

[68] The ISSI Research Group. The ISSI Research Group.

[69] JUNG the Java Universal Network/Graph Framework. Jung - the java universal network/graph framework. `http://jung.sourceforge.net/`.

[70] The Maude System homepage. The Maude Project. `http://maude.cs.uiuc.edu`.

[71] The Maude Team. Maude downloads. `http://maude.cs.uiuc.edu/downloads`.

[72] The MOMENT Project. Directorio de releases de MOMENT. `ftp://moment.dsic.upv.es/releases/`.

[73] The MOMENT Project. Maude Development Tools. `http://moment.dsic.upv.es/mdt`.

[74] The MOMENT Project. Maude for Windows. `http://moment.dsic.upv.es/mfw`.

[75] Adelinde Uhrmacher, Daniela Degenring, and Bernard P. Zeigler. Discrete event multi-level models for systems biology. *T. Comp. Sys. Biology*, 3380:66–89, 2005.

[76] Mareike Ziegler. Implementierung eines Transformationsmoduls in Java zur Abbildung von Signaltransduktions-Instanzen auf CPN-Instanzen. Master's thesis, Technische Universität Braunschweig, Braunschweig, Januar 2007.

Apéndice A

Transformación Transpath2Cpn para MOMENT–QVT

```
1   transformation TranspathToCPN(
2       tpDomain:http:\/\/\/es.upv.dsic.issi/moment/intergenomics/transpath,
3       target cpnDomain:http:\/\/\/es.upv.dsic.issi/moment/intergenomics/cpn)
4   {
5
6       // Key declaration
7
8       key Page {name};
9       key Globbox {id};
10      key Cpnet {page,globbox};
11      key Block {idname};
12      key Enumerated {idname};
13      key Product {idname};
14      key ColorSetElement {name};
15      key Trans {id};
16      key Initmark {id};
17      key Mark {initmark,colorSetElement};
18      key Fusion {name};
19      key Arc {trans,place};
20
21
22      // Creation of the root element
23
24      top relation NetworkToCpnet {
25
26          checkonly domain tpDomain nt1:Network {
27              molecules = molec1 : Molecule {},
28              reactions = react1 : Reaction {}
29          };
30
31          enforce domain cpnDomain cn1:Cpnet{
32              page = page1 : Page {
```

```
                    id = GetPathwayName(nt1),
                    name = GetPathwayName(nt1)
                },
                globbox = gb1 : Globbox {
                    id = 'Declarations'
                }
            };
            where {
                ComplexMoleculeToComplexesBlock(molec1,gb1);
                ReactionToGUIElements(react1,cn1,page1);
            }
        }

        // Declaration of the colorsets from complex molecules

        relation ComplexMoleculeToComplexesBlock {

            checkonly domain tpDomain molec1 : Molecule {};

            checkonly domain cpnDomain gb1 : Globbox {};

            enforce domain cpnDomain block1 : Block {
                globbox = gb1,
                idname = 'Complexes'
            };

            when {
                IsComplexMolecule(molec1);
            }
            where {
                ComplexMoleculeToProduct(molec1,gb1,block1);
            }
        }


        relation ComplexMoleculeToProduct {

            checkonly domain tpDomain molec1 : Molecule {
                statesOf = simplemolec1 : Molecule {}
            };

            checkonly domain cpnDomain gb1 : Globbox {};

            checkonly domain cpnDomain block1 : Block {};

            enforce domain cpnDomain prod1 : Product {
                block = block1,
                idname = GetMoleculeType(molec1)
            };

            where {
                SimpleMoleculeToEnumerated(simplemolec1, gb1, prod1);
            }
        }
```

```
87
88
89          relation SimpleMoleculeToEnumerated {
90
91              checkonly domain tpDomain molec1 : Molecule {};
92
93              checkonly domain cpnDomain gb1 : Globbox {};
94
95              checkonly domain cpnDomain prod1 : Product {};
96
97              enforce domain cpnDomain enum1 : Enumerated {
98                  block = block1 : Block {
99                      globbox = gb1,
100                     idname = 'Ressources'
101                 },
102                 idname = GetMoleculeType(molec1),
103                 usedIn = prod1
104             };
105
106             when {
107                 IsSimpleMolecule(molec1);
108             }
109
110             where {
111                 SimpleMoleculeToColorSetElement(molec1,enum1);
112             }
113         }
114
115
116
117
118         relation SimpleMoleculeToColorSetElement {
119
120             moleculeName1: String;
121
122             checkonly domain tpDomain molec1:Molecule {
123                 name = moleculeName1
124             };
125
126             checkonly domain cpnDomain enum1 : Enumerated {};
127
128             enforce domain cpnDomain colorSetElement1 : ColorSetElement {
129                 name = moleculeName1,
130                 simpleColorSet = enum1
131             };
132         }
133
134
135         // Rules to create the GUI elements from a Reaction
136
137         relation ReactionToGUIElements {
138
139             reactionName1 : String;
140
```

```
141             checkonly domain tpDomain react1 : Reaction {
142                 name = reactionName1,
143                 reactantsCoefficient = rc1 : ReactantsCoefficient {},
144                 producesCoefficient = pc1 : ProductsCoefficient {}
145 
146             };
147 
148             checkonly domain cpnDomain cn1:Cpnet{
149                 globbox = globbox1 : Globbox {}
150             };
151 
152             checkonly domain cpnDomain page1 : Page {};
153 
154             enforce domain cpnDomain trans1 : Trans {
155                 id = reactionName1,
156                 text = 'bind',
157                 page = page1
158             };
159             where {
160                 ReactantsCoefficientToPlaces(rc1,cn1,page1,trans1);
161                 ProductsCoefficientToPlaces(pc1,cn1,page1,trans1);
162             }
163         }
164 
165         relation ReactantsCoefficientToPlaces {
166 
167 
168             checkonly domain tpDomain rc1 : ReactantsCoefficient {
169                 reactants = reactantMolecule1 : Molecule {}
170             };
171 
172             checkonly domain cpnDomain cn1 : Cpnet {};
173 
174             checkonly domain cpnDomain page1 : Page {};
175 
176             checkonly domain cpnDomain trans1 : Trans {};
177 
178             where {
179                 ReactantsToPlaces(reactantMolecule1,cn1,page1,trans1);
180             }
181         }
182 
183         relation ReactantsToPlaces {
184 
185             molecName1 : String;
186             transId1 : String;
187 
188             checkonly domain tpDomain reactantMolecule1 : Molecule {
189                 name = molecName1
190             };
191 
192             checkonly domain cpnDomain cn1 : Cpnet{};
193 
194             checkonly domain cpnDomain page1 : Page {};
```

```
195
196             checkonly domain cpnDomain trans1 : Trans {
197                 id = transId1
198             };
199
200             enforce domain cpnDomain place1 : Place {
201                 id = GetMoleculeType(reactantMolecule1),
202                 page = page1,
203                 fusion = fusion1 : Fusion {
204                     id = GetMoleculeType(reactantMolecule1),
205                     name = GetMoleculeType(reactantMolecule1),
206                     cpnet = cn1
207                 },
208                 initmark = imark1 : Initmark {
209                     id = molecName1
210                 }
211             };
212
213             where {
214                 ReactantsToMarks(reactantMolecule1,page1,trans1,place1,imark1);
215                 ReactantsToArcs(reactantMolecule1,page1,trans1,place1,transId1);
216             }
217         }
218
219         relation ReactantsToMarks {
220
221             molecName1 : String;
222
223             checkonly domain tpDomain reactantMolecule1 : Molecule {
224                 name = molecName1
225             };
226
227             checkonly domain cpnDomain page1 : Page {};
228
229             checkonly domain cpnDomain trans1 : Trans {};
230
231             checkonly domain cpnDomain place1 : Place {};
232
233             checkonly domain cpnDomain imark1 : Initmark {};
234
235             enforce domain cpnDomain mrk1 : Mark {
236                 value = 1,
237                 initmark = imark1/*,
238                 colorSetElement = colorSetElement1 : ColorSetElement {
239                     name = molecName1
240                 }*/
241             };
242         }
243
244         relation ProductsCoefficientToPlaces {
245
246             checkonly domain tpDomain pc1 : ProductsCoefficient {
247                 produces = productMolecule1 : Molecule {}
248             };
```

```
249
250             checkonly domain cpnDomain cn1 : Cpnet{};
251
252             checkonly domain cpnDomain page1 : Page {};
253
254             checkonly domain cpnDomain trans1 : Trans {};
255
256
257             where {
258                 ProductsToPlaces(productMolecule1,cn1,page1,trans1);
259             }
260
261         }
262
263         relation ProductsToPlaces {
264
265             transId1 : String;
266
267             checkonly domain tpDomain productMolecule1 : Molecule {};
268
269             checkonly domain cpnDomain cn1 : Cpnet{};
270
271             checkonly domain cpnDomain page1 : Page {};
272
273             checkonly domain cpnDomain trans1 : Trans {
274                 id = transId1
275             };
276
277             enforce domain cpnDomain place1 : Place {
278                 id = GetMoleculeType(productMolecule1),
279                 // Sólo se genera un place
280                 fusion = fusion1 : Fusion {
281                     id = GetMoleculeType(productMolecule1),
282                     name = GetMoleculeType(productMolecule1),
283                     cpnet = cn1
284                 },
285                 page = page1
286             };
287
288
289             where {
290                 ProductsToArcs(productMolecule1,page1,trans1,place1,transId1);
291             }
292
293         }
294
295
296         relation ReactantsToArcs {
297
298             reactantMolecName1 : String;
299             transId1 : String;
300
301
302             checkonly domain tpDomain reactantMolecule1 : Molecule {
```

```
303                     name = reactantMolecName1
304             };
305
306             checkonly domain cpnDomain page1 : Page {};
307
308             checkonly domain cpnDomain trans1 : Trans {};
309
310             checkonly domain cpnDomain place1 : Place {};
311
312             primitive domain transId1 : String;
313
314
315             enforce domain cpnDomain arc1 : Arc {
316                 id = '{' + reactantMolecName1 + '} => {' + transId1 + '}',
317                 orientation = 'PtoT',
318                 trans = trans1,
319                 place = place1,
320                 page = page1
321             };
322         }
323
324
325         relation ProductsToArcs {
326
327             productMolecName1 : String;
328             transId1 : String;
329
330             checkonly domain tpDomain productMolecule1 : Molecule {
331                     name = productMolecName1
332             };
333
334             checkonly domain cpnDomain page1 : Page {};
335
336             checkonly domain cpnDomain trans1 : Trans {};
337
338             checkonly domain cpnDomain place1 : Place {};
339
340             primitive domain transId1 : String;
341
342
343             enforce domain cpnDomain arc1 : Arc {
344                 id = '{' + transId1  + '} => {' + productMolecName1 + '}',
345                 orientation = 'TtoP',
346                 trans = trans1,
347                 place = place1,
348                 page = page1
349             };
350         }
351
352
353         // Helper functions
354
355
356         function IsSimpleMolecule(molec:Molecule):Bool
```

```
357         {
358             (molec.statesOf -> isEmpty())
359         }
360 
361         function IsComplexMolecule(molec:Molecule):Bool
362         {
363             (not(molec.statesOf -> isEmpty()))
364         }
365 
366         function GetMoleculeType(molec : Molecule) : String
367         {
368             if (molec.statesOf -> isEmpty())
369             then
370                 if (molec.klass -> includes('adaptor proteins'))
371                 then 'A'
372                 else
373                     if (molec.klass -> includes('receptors'))
374                     then 'R'
375                     else 'O'
376                     endif
377                 endif
378             else
379                 if (molec.name = 'LPS:LBP')
380                 then 'OA'
381                 else
382                     if (molec.name = 'LPS:LBP:CD14')
383                     then 'OAR'
384                     else
385                         if
386                         (molec.name = 'ST2:TIRAP' or molec.name = 'ST2:MyD88')
387                         then 'RA'
388                         else 'Unknown' // No debería llegar aquí
389                         endif
390                     endif
391                 endif
392             endif
393 
394         }
395 
396         function GetPathwayName(nw:Network):String
397         {
398             nw.pathway -> asOrderedSet() -> first().name
399         }
400     }
```

Listado A.1: Transformación Transpath2CPN completa en QVT-Relations para MOMENT–QVT

Apéndice B

Transformación Transpath2Cpn para MediniQVT

```
transformation TranspathToCPN(tpDomain:transpath, cpnDomain:cpn)
{

  // Key declaration

  key Page {name};
  key Globbox {id};
  key Cpnet {page,globbox};
  key Block {idname};
  key Enumerated {idname};
  key Product {idname};
  key ColorSetElement {name};
  key Place {id};
  key Trans {id};
  key Initmark {id};
  key Mark {initmark,colorSetElement};
  key Fusion {name};
  key Arc {trans,place};


  /************************************************
   ** Root elements
   ************************************************/

  top relation NetworkToCpnet {

    checkonly domain tpDomain network : transpath::Network {
      molecules = molecule : transpath::Molecule {},
      reactions = reaction : transpath::Reaction {}
    };

    enforce domain cpnDomain cpnet : cpn::Cpnet{
```

```
33            page = page : Page {
34              id = network.pathway.id.first(),
35              name = network.pathway.name.first(),
36              posx = 200,
37              posy = 100,
38              width = 1000,
39              height = 800
40            },
41            globbox = globbox : cpn::Globbox {
42              id = 'Declarations'
43            }
44          };
45          where {
46            ComplexMoleculeToProduct(molecule, globbox);
47            ReactionToGUIElements(reaction, page);
48            //page.performLayout(600, 400, 1000);
49          }
50        }
51
52        /************************************************
53         ** Declarations
54         ************************************************/
55
56        relation ComplexMoleculeToProduct {
57
58          checkonly domain tpDomain molecule : transpath::Molecule {
59            statesOf = simpleMolecule : transpath::Molecule {}
60          };
61
62          enforce domain cpnDomain globbox : cpn::Globbox {
63            declarations = resourcesBlock : cpn::Block {
64              id = 'Resources',
65              idname = 'Resources'
66            },
67            declarations = complexesBlock : cpn::Block {
68              id = 'Complexes',
69              idname = 'Complexes',
70              declarations = product : cpn::Product {
71                idname = GetMoleculeType(molecule)
72              }
73            }
74          };
75          when {
76            IsComplexMolecule(molecule);
77          }
78          where {
79            SimpleMoleculeToEnumerated(simpleMolecule, resourcesBlock, product);
80          }
81        }
82
83        relation SimpleMoleculeToEnumerated {
84
85          checkonly domain tpDomain molecule : transpath::Molecule {
86          };
```

```
87
88            enforce domain cpnDomain resourcesBlock : cpn::Block {
89              declarations = enumerated : cpn::Enumerated {
90                idname = GetMoleculeType(molecule),
91                id = GetMoleculeType(molecule),
92                //usedIn = usedIn->including(product),
93                colorElements = element : cpn::ColorSetElement {
94                  name = molecule.name
95                }
96              }
97            };
98            enforce domain cpnDomain product : cpn::Product {
99              simpleColors =
100               simpleColors
101               ->including(enumerated)
102               ->sortedBy(colorSet : ColorSet | colorSet.idname)
103           };
104
105           when {
106             IsSimpleMolecule(molecule);
107           }
108         }
109
110         /************************************************
111          ** Graphical elements
112          ************************************************/
113
114
115         relation ReactionToGUIElements {
116
117           checkonly domain tpDomain reaction : transpath::Reaction {
118             reactantsCoefficient = reactantsCoefficients : transpath::ReactantsCoefficient
                    {},
119             producesCoefficient = productsCoefficients : transpath::ProductsCoefficient {}
120           };
121
122           enforce domain cpnDomain page : cpn::Page {
123             transs = transition : cpn::Trans {
124               id = reaction.name,
125               text = 'bind ' + BuildTransitionText(reaction),
126               width = 60,
127               hight = 40,
128               fillColour = 'White',
129               fillPattern = 'Solid',
130               fillFilled = false,
131               lineThick = 1,
132               lineType = 'Solid',
133               lineColour = 'Black'
134             }
135           };
136
137           where {
138             ReactantToPlaceArc(reactantsCoefficients.reactants, transition, reaction.name,
                    page);
```

```
139             ProductToPlaceArc(productsCoefficients.produces, transition, reaction.name,
                    page);
140           }
141         }
142
143         relation ReactantToPlaceArc {
144
145           checkonly domain tpDomain reactant : Molecule {};
146
147           checkonly domain cpnDomain trans : cpn::Trans {};
148
149           primitive domain reactionName : String;
150
151           enforce domain cpnDomain page : cpn::Page {
152             places = place : cpn::Place {
153               id = GetMoleculeType(reactant),
154               lineColour =
155                 if Reaction.allInstances().producesCoefficient.produces->includes(reactant)
                        then
156                   'Black'
157                 else
158                   'Lime'
159                 endif
160             },
161             arcs = arc : cpn::Arc {
162               id = '{' + reactant.name + '} => {' + reactionName + '}',
163               orientation = 'PtoT',
164               trans = trans,
165               place = place
166             }
167           };
168           where {
169             FillCommonAttributesInPlaces(place);
170             MoleculeToArcAnnot(reactant,arc);
171             SimpleMoleculeToPlaceType(reactant, place);
172             ComplexMoleculeToPlaceType(reactant, place);
173             SimpleReactantToInitMark(reactant,place);
174           }
175         }
176
177         relation ProductToPlaceArc {
178
179           checkonly domain tpDomain product : Molecule {};
180
181           checkonly domain cpnDomain trans : cpn::Trans {};
182
183           primitive domain reactionName : String;
184
185           enforce domain cpnDomain page : cpn::Page {
186             places = place : cpn::Place {
187               lineColour =
188                 if not(Reaction.allInstances().reactantsCoefficient.reactants->includes(
                        product)) then
189                   'Maroon'
```

```
190                 else
191                   'Black'
192                 endif,
193               id =
194                 if Reaction.allInstances().reactantsCoefficient.reactants->includes(product
                        ) then
195                    GetMoleculeType(product)
196                 else
197                   product.name
198                 endif,
199               text =
200                 if not(Reaction.allInstances().reactantsCoefficient.reactants->includes(
                        product)) then
201                    'Dead end'
202                 else
203                   ''
204                 endif
205             },
206             arcs = arc : cpn::Arc {
207               id = '{' + reactionName + '} => {' + product.name + '}',
208               orientation = 'TtoP',
209               trans = trans,
210               place = place
211             }
212           };
213
214           where {
215             FillCommonAttributesInPlaces(place);
216             MoleculeToArcAnnot(product,arc);
217             SimpleMoleculeToPlaceType(product, place);
218             ComplexMoleculeToPlaceType(product, place);
219           }
220         }
221
222         relation FillCommonAttributesInPlaces {
223
224           enforce domain cpnDomain place : cpn::Place {
225               width = 60,
226               hight = 40,
227               fillColour = 'White',
228               fillPattern = 'Solid',
229               fillFilled = false,
230               lineThick = 1,
231               lineType = 'Solid'
232           };
233
234         }
235
236         relation SimpleReactantToInitMark {
237
238           checkonly domain tpDomain reactant : transpath::Molecule  {
239           };
240
241           enforce domain cpnDomain place : cpn::Place  {
```

```
242             initmark = imark : cpn::Initmark {
243               fillColour = 'White',
244               fillPattern = 'Solid',
245               fillFilled = false,
246               lineThick = 1,
247               lineType = 'Solid',
248               lineColour = 'Lime',
249
250               id = GetMoleculeType(reactant),
251               marks = mark : cpn::Mark {
252                 value =
253                   --- This value is not a valid value, it is set in this way in order to
254                   --- obtain a valid CPNet. This marking must be corrected by biologists.
255                   --- Now, we initialize this value to the number of molecules of each kind
256                   --- that are involved in the reactions of the pathway
257                   reactant.rkoutsCoefficient.coefficient->sum(),
258                 colorSetElement = colorSetElement : cpn::ColorSetElement {
259                   name = reactant.name
260                 }
261               }
262             }
263           };
264
265           when {
266             IsSimpleMolecule(reactant);
267           }
268         }
269
270         relation SimpleMoleculeToPlaceType {
271
272           checkonly domain tpDomain molecule : Molecule {};
273
274           enforce domain cpnDomain place : Place {
275             type = enumerated : cpn::Enumerated {
276               idname = GetMoleculeType(molecule)
277             }
278           };
279           when {
280             IsSimpleMolecule(molecule);
281           }
282         }
283
284         relation ComplexMoleculeToPlaceType {
285
286           checkonly domain tpDomain molecule : Molecule {};
287
288           enforce domain cpnDomain place : Place {
289             type = product : cpn::Product {
290               idname = GetMoleculeType(molecule)
291             }
292           };
293           when {
294             IsComplexMolecule(molecule);
295           }
```

```
296         }
297
298         relation MoleculeToArcAnnot {
299
300           checkonly domain tpDomain molecule : Molecule {};
301
302           enforce domain cpnDomain arc : cpn::Arc {
303             headsize = 1,
304             currentcyckle = '2',
305             fillColour = 'White',
306             fillPattern = 'Solid',
307             fillFilled = false,
308             lineThick = 1,
309             lineType = 'Solid',
310             lineColour =
311               if IsSimpleMolecule(molecule) then
312                 'Lime'
313               else
314                 'Black'
315               endif,
316             annot = annot : cpn::Annot {
317               fillColour = 'White',
318               fillPattern = 'Solid',
319               fillFilled = false,
320               lineThick = 1,
321               lineType = 'Solid',
322               lineColour =
323               if IsSimpleMolecule(molecule) then
324                 'Lime'
325               else
326                 'Black'
327               endif,
328               text =
329               if IsSimpleMolecule(molecule) then
330                 '1`' + molecule.name
331               else
332                 '(' + BuildMoleculeComponentsList(molecule) + ')'
333               endif
334             }
335           };
336
337         }
338
339         /************************************************
340          ** Helper functions
341          ************************************************/
342
343
344         query IsSimpleMolecule(molec:Molecule):Boolean
345         {
346           (molec.statesOf -> isEmpty())
347         }
348
349         query IsComplexMolecule(molec:Molecule):Boolean
```

```
350         {
351           (not(molec.statesOf -> isEmpty()))
352         }
353
354         query GetMoleculeType(molecule : transpath::Molecule) : String
355         {
356           if IsSimpleMolecule(molecule) then
357             GetSimpleMoleculeType(molecule)
358           else
359             GetComplexMoleculeType(molecule)
360           endif
361         }
362
363         query GetSimpleMoleculeType(molec : Molecule) : String
364         {
365           if (molec.klass -> includes('adaptor proteins'))
366           then 'A'
367           else
368             if (molec.klass -> includes('receptors'))
369             then 'R'
370             else 'O'
371             endif
372           endif
373         }
374
375         query GetComplexMoleculeType(complexMolecule : Molecule) : String
376         {
377           complexMolecule.statesOf
378           ->collect(m : Molecule | GetSimpleMoleculeType(m))
379           ->asSet()
380           ->sortedBy(type : String | type )
381           ->iterate(type : String; complexBlockID : String = '' | complexBlockID.concat(
                  type))
382         }
383
384         query BuildTransitionText(reaction : Reaction) : String
385         {
386           let moleculeNames : Sequence(String) =
387             reaction.reactantsCoefficient.reactants.name
388           in
389             moleculeNames
390             ->excluding(moleculeNames.last())
391             ->iterate(moleculeName : String; text : String = '' | text.concat(moleculeName)
                      .concat(','))
392             .concat(moleculeNames.last())
393         }
394         query BuildMoleculeComponentsList(molecule : Molecule) : String
395         {
396
397           let moleculesSet : OrderedSet(Molecule) =
398             molecule.statesOf
399             ->asSet()
400             ->sortedBy(molecule : Molecule | GetMoleculeType(molecule))
401           in
```

```
402             moleculesSet
403             ->excluding(moleculesSet.last())
404             ->iterate(molecule : Molecule; text : String = '' | text.concat(molecule.name).
                    concat(','))
405             .concat(moleculesSet.last().name)
406         }
407       }
```

Listado B.1: Transformación Transpath2CPN completa en QVT-Relations para MediniQVT

Apéndice C

DTD de la base de datos TRANSPATH®

```
<!-- TRANSPATH PROFESSIONAL DATABASE DTD -->
<!-- Authors: Frank Schacherer, Mathias Krull (contact: mkl@biobase.de), Nico Voss (nvo@biobase.de) -->
<!-- copy: copyright (C), Biobase GmbH -->
<!ENTITY % classes "Molecule | Reaction | Gene | Reference | Annotate | Pathway">
<!ELEMENT network (%classes;)*> <!--root element-->
<!ATTLIST network
  extent CDATA #REQUIRED
  source CDATA #REQUIRED
  version CDATA #REQUIRED
        xmlns:xlink CDATA    #FIXED "http://www.w3.org/1999/xlink"
>
<!-- elements for common declared fields of all classes -->
<!ENTITY % entry-fields "extid?, secid?, creator?, updator?">
<!ELEMENT extid (#PCDATA)>
<!ELEMENT secid (#PCDATA)>
<!ELEMENT creator (#PCDATA)>
<!ELEMENT updator (#PCDATA)>
<!ELEMENT accnos (#PCDATA)>
<!ELEMENT genomic_accnos (#PCDATA)>
<!-- elements for common declared fields of classes "Molecule", "Reaction", and "Gene" and "Pathway"  -->
<!ELEMENT name (#PCDATA)>
<!ELEMENT references (item)*>
<!ELEMENT members (item)*>
<!ELEMENT groups (item)*>
<!-- elements for common declared fields of classes "Molecule" and "Reaction"  -->
<!ELEMENT locations (go?, expression_level?, negative?, tissue?, cytomer_organ?,
    cytomer_cell?, cytomer_stage?, cytomer_species?, cytomer_comment?, cellline?,
    stage?, compartment?, species*, accnos?, method?, material?, references?)*>
<!ELEMENT expression_level (#PCDATA)>
<!ELEMENT negative (#PCDATA)>
```

```
29      <!ELEMENT cytomer_organ (#PCDATA)>
30      <!ELEMENT cytomer_cell (#PCDATA)>
31      <!ELEMENT cytomer_stage (#PCDATA)>
32      <!ELEMENT cytomer_species (#PCDATA)>
33      <!ELEMENT cytomer_comment (#PCDATA)>
34      <!ELEMENT method (#PCDATA)>
35      <!ELEMENT material (#PCDATA)>
36      <!ELEMENT tissue (#PCDATA)>
37      <!ELEMENT compartment (#PCDATA)>
38      <!ELEMENT cellline (#PCDATA)>
39      <!ELEMENT species (#PCDATA)>
40      <!ELEMENT stage (#PCDATA)>
41      <!ELEMENT components (item)*>
42      <!ELEMENT pathways (item)*>
43      <!ELEMENT semantic (item)*>
44      <!ELEMENT pathway_step (item)*>
45      <!ELEMENT molecular_evidence (item)*>
46      <!ELEMENT composition (item)*>
47      <!ELEMENT decomposition (item)*>
48      <!-- elements for common declared fields of classes "Molecule" and "Gene"  -->
49      <!ELEMENT rkins (item)*> <!--downstream reactions-->
50      <!ELEMENT rkouts (item)*> <!--upstream reactions-->
51      <!ELEMENT fullname (#PCDATA)>
52      <!-- elements for common declared fields of classes "Molecule" and "Gene" and "
            Pathway" -->
53      <!ELEMENT synonyms (#PCDATA)>
54      <!-- elements for common declared fields of classes "Molecule" and "Reaction" and "
            Gene" and "Pathway" -->
55      <!ELEMENT type (#PCDATA)>
56      <!ELEMENT comments (item)*>
57      <!--class Annotate and its declared fields -->
58      <!ELEMENT Annotate (%entry-fields;, category?, text, source?, accnos?)>
59      <!ATTLIST Annotate
60        id ID #REQUIRED
61      >
62      <!ELEMENT category (#PCDATA)>
63      <!ELEMENT text (#PCDATA)>
64      <!ELEMENT source (item)*>
65      <!--class Molecule  and its declared fields -->
66      <!ELEMENT Molecule (%entry-fields;, factor?, type?, quality?, name, synonyms*,
            firstpos?, lastpos?, klass?, seq_source?, seq_len?, seq_mw?, exp_mw?, seq_ip?,
            exp_ip?, sequence?, species?, accnos*, genomic_accnos*, comments?, references?,
            members?, groups?, locations*, go*, states?, stateofs?, rkins?, rkouts?,
            catalyzes?, inhibits?, parts?, bulks?, pathways?)>
67      <!ATTLIST Molecule
68        id ID #REQUIRED
69      >
70      <!ELEMENT klass (#PCDATA)> <!--classification-->
71      <!ELEMENT firstpos (#PCDATA)> <!--starting position of feature/motif-->
72      <!ELEMENT lastpos (#PCDATA)> <!--end position of feature/motif-->
73      <!ELEMENT seq_source (#PCDATA)> <!--sequence source-->
74      <!ELEMENT seq_len (#PCDATA)> <!--sequence length-->
75      <!ELEMENT seq_mw (#PCDATA)> <!--calculated molecular weight-->
76      <!ELEMENT exp_mw (#PCDATA)> <!--experimental molecular weight-->
```

```
77      <!ELEMENT seq_ip (#PCDATA)> <!--calculated isoelectric point-->
78      <!ELEMENT exp_ip (#PCDATA)> <!--experimental isoelectric point-->
79      <!ELEMENT sequence (#PCDATA)>
80      <!ELEMENT gene_acc (#PCDATA)> <!--gene_acc in TRANSFAC-->
81      <!ELEMENT factor (#PCDATA)> <!--factor_no in TRANSFAC-->
82      <!ELEMENT go (#PCDATA)> <!--gene ontology-->
83      <!ELEMENT states (item)*> <!--modified forms-->
84      <!ELEMENT stateofs (item)*> <!--modified form of-->
85      <!ELEMENT catalyzes (item)*> <!--catalyzed reactions-->
86      <!ELEMENT inhibits (item)*> <!--inhibited or modulated reactions-->
87      <!ELEMENT parts (item)*> <!--motifs-->
88      <!ELEMENT bulks (item)*> <!--motif of-->
89      <!--class Reaction  and its declared fields -->
90      <!ELEMENT Reaction (%entry-fields;, type?, quality?, name, effect?, reversible?,
            accnos*, comments?, references?, locations*, semantic?, pathway_step?,
            molecular_evidence?, decomposition?, composition?, pathways?, reactants?,
            produces?, enzyme?, inhibitor?)>
91      <!ATTLIST Reaction
92        id ID #REQUIRED
93      >
94      <!ELEMENT quality (#PCDATA)>
95      <!ELEMENT produces (item, stoichiometry?)*>
96      <!ELEMENT reactants (item, stoichiometry?)*>
97      <!ELEMENT stoichiometry (#PCDATA)>
98      <!ELEMENT enzyme (item)?>
99      <!ELEMENT inhibitor (item)*>
100     <!ELEMENT effect (#PCDATA)>
101     <!ELEMENT reversible (#PCDATA)>
102     <!--class Reference   and its declared fields -->
103     <!ELEMENT Reference (%entry-fields;, title?, publication?, authors?, accnos*,
            components?)>
104     <!ATTLIST Reference
105       id ID #REQUIRED
106     >
107     <!ELEMENT title (#PCDATA)>
108     <!ELEMENT publication (#PCDATA)>
109     <!ELEMENT authors (#PCDATA)>
110     <!--class Gene and its declared fields -->
111     <!ELEMENT Gene (%entry-fields;, name, fullname?, synonyms*, species?, accnos*,
            references?, members?, groups?, rkins?, rkouts?)>
112     <!ATTLIST Gene
113       id ID #REQUIRED
114     >
115     <!--class Pathway and its declared fields -->
116     <!ELEMENT Pathway (%entry-fields;, name, synonyms*, type?, chains?, pathways?,
            description?, comments?, reactions_involved?, molecules_involved?, references?)>
117     <!ATTLIST Pathway
118       id ID #REQUIRED
119     >
120     <!ELEMENT chains (item)*>
121     <!ELEMENT reactions_involved (item)*>
122     <!ELEMENT molecules_involved (item)*>
123     <!ELEMENT description (#PCDATA)>
124
```

```
125       <!-- item provides links to other actual database entries using XLink -->
126       <!ELEMENT item (#PCDATA)>
127       <!ATTLIST item
128         type (%classes;) #REQUIRED
129         xlink:type (simple) #FIXED "simple"
130         xlink:href CDATA #REQUIRED
131         xlink:show (new | replace) "new"
132         xlink:actuate (onRequest) #FIXED "onRequest"
133       >
```

Listado C.1: DTD de la base de datos TRANSPATH®

Apéndice D

Representación en XML del *pathway* del TLR4

```
<?xml version="1.0" encoding="UTF-8"?>
<network extent="pathway" source="bbdb101" version="7.1" xmlns:xlink="http://www.w3.org/1999/xlink">
    <Pathway id="CH000000755">
        <!-- Copyright (c) Biobase GmbH -->
        <creator>mkl</creator>
        <updator>mkl</updator>
        <name>TLR4 pathway</name>
        <synonyms>LPS pathway; lipopolysaccharide/endotoxin pathway; .
        </synonyms>
        <type>pathway</type>
        <chains>
            <PathwayReference id="CH000000569"/>
            <PathwayReference id="CH000000571"/>
            <PathwayReference id="CH000000572"/>
            <PathwayReference id="CH000000831"/>
        </chains>
        <pathways>
        </pathways>
        <comments>
        </comments>
        <reactions_involved>
        </reactions_involved>
        <molecules_involved>
        </molecules_involved>
        <references>
        </references>
    </Pathway>

    <Pathway id="CH000000569">
        <!-- Copyright (c) Biobase GmbH -->
        <creator>mkl</creator>
```

```
32              <updator>mkl</updator>
33              <name>LPS ---&gt; NF-kappaB</name>
34              <synonyms>.
35              </synonyms>
36              <type>chain</type>
37              <chains>
38              </chains>
39              <pathways>
40               </pathways>
41              <comments>
42              </comments>
43              <reactions_involved>
44                  <ReactionReference id="XN000023302"/>
45                  <ReactionReference id="XN000023303"/>
46                  <ReactionReference id="XN000032619"/>
47              </reactions_involved>
48              <molecules_involved>
49              </molecules_involved>
50              <references>
51              </references>
52          </Pathway>
53
54          <Pathway id="CH000000571">
55              <!-- Copyright (c) Biobase GmbH -->
56              <creator>cch</creator>
57              <updator>mkl</updator>
58              <name>LPS ---&gt; JNK</name>
59              <synonyms>.
60              </synonyms>
61              <type>chain</type>
62              <chains>
63              </chains>
64              <pathways>
65              </pathways>
66              <comments>
67              </comments>
68              <reactions_involved>
69                  <ReactionReference id="XN000023302"/>
70                  <ReactionReference id="XN000023303"/>
71              </reactions_involved>
72              <molecules_involved>
73              </molecules_involved>
74              <references>
75              </references>
76          </Pathway>
77
78          <Pathway id="CH000000572">
79              <!-- Copyright (c) Biobase GmbH -->
80              <creator>cch</creator>
81              <updator>mkl</updator>
82              <name>LPS ---&gt; p38</name>
83              <synonyms>.
84              </synonyms>
85              <type>chain</type>
```

```
86              <chains>
87              </chains>
88              <pathways>
89              </pathways>
90              <comments>
91              </comments>
92              <reactions_involved>
93                  <ReactionReference id="XN000023302"/>
94                  <ReactionReference id="XN000023303"/>
95              </reactions_involved>
96              <molecules_involved>
97              </molecules_involved>
98              <references>
99              </references>
100         </Pathway>
101
102         <Pathway id="CH000000831">
103             <!-- Copyright (c) Biobase GmbH -->
104             <creator>mkl</creator>
105             <updator>mkl</updator>
106             <name>LPS ---&gt; IRF-3:IRF-7:CBP:p300</name>
107             <synonyms>.
108             </synonyms>
109             <type>chain</type>
110             <chains>
111             </chains>
112             <pathways>
113             </pathways>
114             <comments>
115             </comments>
116             <reactions_involved>
117                 <ReactionReference id="XN000032621"/>
118             </reactions_involved>
119             <molecules_involved>
120             </molecules_involved>
121             <references>
122             </references>
123         </Pathway>
124
125         <Reaction id="XN000023302">
126             <!-- Copyright (c) Biobase GmbH -->
127             <creator>cch</creator>
128             <updator>cch</updator>
129             <type>pathway step</type>
130             <name>LPS + LBP &lt;=&gt; LPS:LBP</name>
131             <effect>binding</effect>
132             <reversible>true</reversible>
133             <direct>direct</direct>
134             <comments>
135             </comments>
136             <references>
137             </references>
138             <semantic>
139             </semantic>
```

```
140             <pathway_step>
141             </pathway_step>
142             <molecular_evidence>
143             </molecular_evidence>
144             <decomposition>
145             </decomposition>
146             <composition>
147             </composition>
148             <pathways>
149             </pathways>
150             <reactants>
151                 <MoleculeReference id="MO000016882"/>
152                 <MoleculeReference id="MO000019420"/>
153             </reactants>
154             <produces>
155                 <MoleculeReference id="MO000021928"/>
156             </produces>
157             <enzyme>
158             </enzyme>
159             <inhibitor>
160             </inhibitor>
161         </Reaction>
162 
163         <Reaction id="XN000023303">
164             <!-- Copyright (c) Biobase GmbH -->
165             <creator>cch</creator>
166             <updator>cch</updator>
167             <type>pathway step</type>
168             <name>LPS:LBP + CD14 &lt;=&gt; LPS:LBP:CD14</name>
169             <effect>binding</effect>
170             <reversible>true</reversible>
171             <direct>direct</direct>
172             <comments>
173             </comments>
174             <references>
175             </references>
176             <semantic>
177             </semantic>
178             <pathway_step>
179             </pathway_step>
180             <molecular_evidence>
181              </molecular_evidence>
182             <decomposition>
183             </decomposition>
184             <composition>
185             </composition>
186             <pathways>
187             </pathways>
188             <reactants>
189                 <MoleculeReference id="MO000018132"/>
190                 <MoleculeReference id="MO000021928"/>
191             </reactants>
192             <produces>
193                 <MoleculeReference id="MO000021929"/>
```

```
194                 </produces>
195                 <enzyme>
196                 </enzyme>
197                 <inhibitor>
198                 </inhibitor>
199             </Reaction>
200
201             <Reaction id="XN0000032621">
202                 <!-- Copyright (c) Biobase GmbH -->
203                 <creator>mkl</creator>
204                 <updator>mkl</updator>
205                 <type>pathway step</type>
206                 <name>ST2 + TIRAP &lt;=&gt; ST2:TIRAP</name>
207                 <effect>binding</effect>
208                 <reversible>true</reversible>
209                 <direct>direct</direct>
210                 <comments>
211                 </comments>
212                 <references>
213                 </references>
214                 <semantic>
215                 </semantic>
216                 <pathway_step>
217                 </pathway_step>
218                 <molecular_evidence>
219                 </molecular_evidence>
220                 <decomposition>
221                 </decomposition>
222                 <composition>
223                 </composition>
224                 <pathways>
225                 </pathways>
226                 <reactants>
227                     <MoleculeReference id="M0000022528"/>
228                     <MoleculeReference id="M0000044786"/>
229                 </reactants>
230                 <produces>
231                     <MoleculeReference id="M0000044801"/>
232                 </produces>
233                 <enzyme>
234                     <ReactionEnzymeRef id= "M0000044801"/>
235                 </enzyme>
236                 <inhibitor>
237                 </inhibitor>
238             </Reaction>
239
240             <Reaction id="XN0000032619">
241                 <!-- Copyright (c) Biobase GmbH -->
242                 <creator>mkl</creator>
243                 <updator>mkl</updator>
244                 <type>pathway step</type>
245                 <name>ST2 + MyD88 &lt;=&gt; ST2:MyD88</name>
246                 <effect>binding</effect>
247                 <reversible>true</reversible>
```

```
248             <direct>direct</direct>
249             <comments>
250             </comments>
251             <references>
252              </references>
253             <semantic>
254             </semantic>
255             <pathway_step>
256             </pathway_step>
257             <molecular_evidence>
258             </molecular_evidence>
259             <decomposition>
260             </decomposition>
261             <composition>
262             </composition>
263             <pathways>
264             </pathways>
265             <reactants>
266                 <MoleculeReference id="MO000016573"/>
267                 <MoleculeReference id="MO000044786"/>
268             </reactants>
269             <produces>
270                 <MoleculeReference id="MO000044800"/>
271              </produces>
272             <enzyme>
273             </enzyme>
274             <inhibitor>
275             </inhibitor>
276         </Reaction>
277
278
279
280         <Molecule id="MO000016882">
281             <!-- Copyright (c) Biobase GmbH -->
282             <creator>ulg</creator>
283             <updator>mkl</updator>
284             <type>other</type>
285             <name>LPS</name>
286             <synonyms>endotoxin</synonyms>
287             <synonyms>lipopolysaccharide</synonyms>
288             <comments>
289             </comments>
290             <references>
291             </references>
292             <members>
293             </members>
294             <groups>
295             </groups>
296             <states>
297             </states>
298             <stateofs>
299             </stateofs>
300             <rkins>
301             </rkins>
```

```
302                 <rkouts>
303                 </rkouts>
304                 <catalyzes>
305                 </catalyzes>
306                 <inhibits>
307                 </inhibits>
308                 <pathways>
309                 </pathways>
310             </Molecule>
311
312             <Molecule id="MO000019420">
313                 <!-- Copyright (c) Biobase GmbH -->
314                 <secid>,  </secid>
315                 <creator>ulg</creator>
316                 <updator>cch</updator>
317                 <type>orthogroup</type>
318                 <name>LBP</name>
319                 <synonyms>lipopolysacharide-binding protein</synonyms>
320                 <klass>adaptor proteins</klass>
321                 <accnos>InterPro:IPR001124; Lipid-binding serum glycoprotein. [
                        :::interpro_reference:::]</accnos>
322                 <comments>
323                 </comments>
324                 <references>
325                 </references>
326                 <members>
327                 </members>
328                 <groups>
329                 </groups>
330                 <states>
331                 </states>
332                 <stateofs>
333                 </stateofs>
334                 <rkins>
335                 </rkins>
336                 <rkouts>
337                 </rkouts>
338                 <catalyzes>
339                 </catalyzes>
340                 <inhibits>
341                 </inhibits>
342                 <pathways>
343                 </pathways>
344             </Molecule>
345
346             <Molecule id="MO000021928">
347                 <!-- Copyright (c) Biobase GmbH -->
348                 <creator>mkl</creator>
349                 <updator>cch</updator>
350                 <type>orthocomplex</type>
351                 <name>LPS:LBP</name>
352                 <comments>
353                 </comments>
354                 <references>
```

```
355             </references>
356             <members>
357             </members>
358             <groups>
359             </groups>
360             <states>
361             </states>
362             <stateofs>
363                 <MoleculeReference id="MO000016882"/>
364                 <MoleculeReference id="MO000019420"/>
365             </stateofs>
366             <rkins>
367             </rkins>
368             <rkouts>
369             </rkouts>
370             <catalyzes>
371             </catalyzes>
372             <inhibits>
373             </inhibits>
374             <pathways>
375             </pathways>
376         </Molecule>
377
378         <Molecule id="MO000018132">
379             <!-- Copyright (c) Biobase GmbH -->
380             <creator>cch</creator>
381             <updator>spi</updator>
382             <type>orthogroup</type>
383             <name>CD14</name>
384             <synonyms>gp55</synonyms>
385             <synonyms>LBP-receptor</synonyms>
386             <synonyms>LPS receptor</synonyms>
387             <synonyms>LPS-R</synonyms>
388             <klass>membrane-transducing components; receptors</klass>
389             <accnos>InterPro:IPR001611; Leucine-rich repeat. [:::interpro_reference:::]</
                    accnos>
390             <comments>
391             </comments>
392             <references>
393             </references>
394             <members>
395             </members>
396             <groups>
397             </groups>
398             <states>
399             </states>
400             <stateofs>
401             </stateofs>
402             <rkins>
403             </rkins>
404             <rkouts>
405             </rkouts>
406             <catalyzes>
407             </catalyzes>
```

```
408             <inhibits>
409             </inhibits>
410             <pathways>
411             </pathways>
412         </Molecule>
413
414         <Molecule id="MO0000021929">
415             <!-- Copyright (c) Biobase GmbH -->
416             <creator>mkl</creator>
417             <updator>mkl</updator>
418             <type>orthocomplex</type>
419             <name>LPS:LBP:CD14</name>
420             <comments>
421             </comments>
422             <references>
423             </references>
424             <members>
425             </members>
426             <groups>
427             </groups>
428             <states>
429             </states>
430             <stateofs>
431                 <MoleculeReference id="MO0000016882"/>
432                 <MoleculeReference id="MO0000018132"/>
433                 <MoleculeReference id="MO0000019420"/>
434             </stateofs>
435             <rkins>
436             </rkins>
437             <rkouts>
438             </rkouts>
439             <catalyzes>
440             </catalyzes>
441             <inhibits>
442             </inhibits>
443             <pathways>
444             </pathways>
445         </Molecule>
446
447         <Molecule id="MO0000022528">
448             <!-- Copyright (c) Biobase GmbH -->
449             <creator>mkl</creator>
450             <updator>mkl</updator>
451             <type>orthogroup</type>
452             <name>TIRAP</name>
453             <synonyms>Toll-interleukin I receptor domain-containing adapter protein</
                    synonyms>
454             <klass>membrane-transducing components; receptors; cytokine receptor family;
                    TIR superfamily | adaptor proteins</klass>
455             <accnos>InterPro:IPR000157; TIR. [:::interpro_reference:::]</accnos>
456             <comments>
457             </comments>
458             <references>
459              </references>
```

```
460             <members>
461             </members>
462             <groups>
463             </groups>
464             <states>
465             </states>
466             <stateofs>
467             </stateofs>
468             <rkins>
469             </rkins>
470             <rkouts>
471             </rkouts>
472             <catalyzes>
473             </catalyzes>
474             <inhibits>
475             </inhibits>
476             <pathways>
477             </pathways>
478         </Molecule>
479
480         <Molecule id="MO0000044786">
481             <!-- Copyright (c) Biobase GmbH -->
482             <creator>mkl</creator>
483             <updator>mkl</updator>
484             <type>orthogroup</type>
485             <name>ST2</name>
486             <synonyms>DER4</synonyms>
487             <synonyms>FIT-1</synonyms>
488             <synonyms>IL1RL1</synonyms>
489             <synonyms>interleukin 1 receptor-like 1</synonyms>
490             <synonyms>MGC32623</synonyms>
491             <synonyms>ST2</synonyms>
492             <synonyms>ST2L</synonyms>
493             <synonyms>ST2V</synonyms>
494             <synonyms>T1</synonyms>
495             <klass>membrane-transducing components; receptors; cytokine receptor family;
                    TIR superfamily</klass>
496             <comments>
497             </comments>
498             <references>
499             </references>
500             <members>
501             </members>
502             <groups>
503             </groups>
504             <states>
505             </states>
506             <stateofs>
507             </stateofs>
508             <rkins>
509             </rkins>
510             <rkouts>
511             </rkouts>
512             <catalyzes>
```

```
513             </catalyzes>
514             <inhibits>
515             </inhibits>
516             <pathways>
517             </pathways>
518         </Molecule>
519
520         <Molecule id="MO0000044801">
521             <!-- Copyright (c) Biobase GmbH -->
522             <creator>mkl</creator>
523             <type>orthocomplex</type>
524             <name>ST2:TIRAP</name>
525             <comments>
526             </comments>
527             <references>
528             </references>
529             <members>
530             </members>
531             <groups>
532             </groups>
533             <states>
534             </states>
535             <stateofs>
536                 <MoleculeReference id="MO0000022528"/>
537                 <MoleculeReference id="MO0000044786"/>
538             </stateofs>
539             <rkins>
540             </rkins>
541             <rkouts>
542             </rkouts>
543             <catalyzes>
544             </catalyzes>
545             <inhibits>
546             </inhibits>
547             <pathways>
548             </pathways>
549         </Molecule>
550
551         <Molecule id="MO0000016573">
552             <!-- Copyright (c) Biobase GmbH -->
553             <creator>ulg</creator>
554             <updator>oke</updator>
555             <type>orthogroup</type>
556             <name>MyD88</name>
557             <synonyms>myeloid differentiation factor 88</synonyms>
558             <klass>adaptor proteins; adaptors for TIR family of receptors</klass>
559             <accnos>InterPro:IPR000157; TIR. [:::interpro_reference:::]</accnos>
560             <accnos>InterPro:IPR004075; Interleukin-1 receptor, type I/Toll precursor. [
                    :::interpro_reference:::]</accnos>
561             <accnos>InterPro:IPR000488; Death domain. [:::interpro_reference:::]</accnos>
562             <comments>
563             </comments>
564             <references>
565             </references>
```

```
566             <members>
567             </members>
568             <groups>
569             </groups>
570             <states>
571             </states>
572             <stateofs>
573             </stateofs>
574             <rkins>
575             </rkins>
576             <rkouts>
577             </rkouts>
578             <catalyzes>
579             </catalyzes>
580             <inhibits>
581             </inhibits>
582             <pathways>
583             </pathways>
584         </Molecule>
585
586         <Molecule id="MO000044800">
587             <!-- Copyright (c) Biobase GmbH -->
588             <creator>mkl</creator>
589             <type>orthocomplex</type>
590             <name>ST2:MyD88</name>
591             <comments>
592             </comments>
593             <references>
594             </references>
595             <members>
596             </members>
597             <groups>
598             </groups>
599             <states>
600             </states>
601             <stateofs>
602                 <MoleculeReference id="MO000016573"/>
603                 <MoleculeReference id="MO000044786"/>
604             </stateofs>
605             <rkins>
606             </rkins>
607             <rkouts>
608             </rkouts>
609             <catalyzes>
610             </catalyzes>
611             <inhibits>
612             </inhibits>
613             <pathways>
614             </pathways>
615         </Molecule>
616
617     </network>
```

Listado D.1: Instancia en XML del pathway del TLR4 extraída de la base de datos TRANSPATH®

Apéndice E

Representación en XMI del *pathway* del TLR4

```
<?xml version="1.0" encoding="ASCII"?>
<transpath:Network xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" xmlns:transpath="http:///es.upv.dsic.
    issi/moment/intergenomics/transpath">
  <molecules id="MO0000016882" name="LPS" rkoutsCoefficient="//@reactions.0/
      @reactantsCoefficient.0">
    <moleculeType>other</moleculeType>
    <synonyms>endotoxin</synonyms>
  </molecules>
  <molecules id="MO0000019420" name="LBP" rkoutsCoefficient="//@reactions.0/
      @reactantsCoefficient.1">
    <moleculeType>orthogroup</moleculeType>
    <klass>adaptor proteins</klass>
    <synonyms>lipopolysacharide-binding protein</synonyms>
  </molecules>
  <molecules id="MO0000021928" name="LPS:LBP" rkinsCoefficient="//@reactions.0/
      @producesCoefficient.0" rkoutsCoefficient="//@reactions.1/@reactantsCoefficient
      .1" statesOf="//@molecules.0 //@molecules.1">
    <moleculeType>orthocomplex</moleculeType>
  </molecules>
  <molecules id="MO0000018132" name="CD14" rkoutsCoefficient="//@reactions.1/
      @reactantsCoefficient.0">
    <moleculeType>orthogroup</moleculeType>
    <klass>membrane-transducing components; receptors</klass>
    <synonyms>gp55</synonyms>
  </molecules>
  <molecules id="MO0000021929" name="LPS:LBP:CD14" rkinsCoefficient="//@reactions.1/
      @producesCoefficient.0" statesOf="//@molecules.0 //@molecules.3 //@molecules.1"
      >
    <moleculeType>orthocomplex</moleculeType>
  </molecules>
```

```
23        <molecules id="MO0000022528" name="TIRAP" rkoutsCoefficient="//@reactions.2/
              @reactantsCoefficient.0">
24          <moleculeType>orthogroup</moleculeType>
25          <klass>membrane-transducing components; receptors; cytokine receptor family; TIR
                superfamily | adaptor proteins</klass>
26          <synonyms>Toll-interleukin I receptor domain-containing adapter protein</synonyms
                >
27        </molecules>
28        <molecules id="MO0000044786" name="ST2" rkoutsCoefficient="//@reactions.2/
              @reactantsCoefficient.1 //@reactions.3/@reactantsCoefficient.1">
29          <moleculeType>orthogroup</moleculeType>
30          <klass>membrane-transducing components; receptors; cytokine receptor family; TIR
                superfamily</klass>
31          <synonyms>DER4</synonyms>
32        </molecules>
33        <molecules id="MO0000044801" name="ST2:TIRAP" rkinsCoefficient="//@reactions.2/
              @producesCoefficient.0" statesOf="//@molecules.5 //@molecules.6">
34          <moleculeType>orthocomplex</moleculeType>
35        </molecules>
36        <molecules id="MO0000016573" name="MyD88" rkoutsCoefficient="//@reactions.3/
              @reactantsCoefficient.0">
37          <moleculeType>orthogroup</moleculeType>
38          <klass>adaptor proteins; adaptors for TIR family of receptors</klass>
39          <synonyms>myeloid differentiation factor 88</synonyms>
40        </molecules>
41        <molecules id="MO0000044800" name="ST2:MyD88" rkinsCoefficient="//@reactions.3/
              @producesCoefficient.0" statesOf="//@molecules.8 //@molecules.6">
42          <moleculeType>orthocomplex</moleculeType>
43        </molecules>
44        <reactions id="XN0000023302" name="LPS + LBP &amp;lt;=> LPS:LBP" reversible="true"
              direct="true">
45          <reactionType xsi:nil="true"/>
46          <effects>binding</effects>
47          <reactantsCoefficient reactants="//@molecules.0"/>
48          <reactantsCoefficient reactants="//@molecules.1"/>
49          <producesCoefficient produces="//@molecules.2"/>
50        </reactions>
51        <reactions id="XN0000023303" name="LPS:LBP + CD14 &amp;lt;=> LPS:LBP:CD14" reversible="
              true" direct="true">
52          <reactionType xsi:nil="true"/>
53          <effects>binding</effects>
54          <reactantsCoefficient reactants="//@molecules.3"/>
55          <reactantsCoefficient reactants="//@molecules.2"/>
56          <producesCoefficient produces="//@molecules.4"/>
57        </reactions>
58        <reactions id="XN0000032621" name="ST2 + TIRAP &amp;lt;=> ST2:TIRAP" reversible="true"
              direct="true">
59          <reactionType xsi:nil="true"/>
60          <effects>binding</effects>
61          <reactantsCoefficient reactants="//@molecules.5"/>
62          <reactantsCoefficient reactants="//@molecules.6"/>
63          <producesCoefficient produces="//@molecules.7"/>
64        </reactions>
```

```
65        <reactions id="XNOOOO32619" name="ST2 + MyD88 &lt;=> ST2:MyD88" reversible="true"
              direct="true">
66          <reactionType xsi:nil="true"/>
67          <effects>binding</effects>
68          <reactantsCoefficient reactants="//@molecules.8"/>
69          <reactantsCoefficient reactants="//@molecules.6"/>
70          <producesCoefficient produces="//@molecules.9"/>
71        </reactions>
72        <pathway id="CHOOOOOO755" name="TLR4 pathway">
73          <synonyms>LPS pathway; lipopolysaccharide/endotoxin pathway; .&#xA;        </
                synonyms>
74          <chains id="CHOOOOOO569" name="LPS ---> NF-kappaB">
75            <synonyms>.&#xA;        </synonyms>
76          </chains>
77          <chains id="CHOOOOOO571" name="LPS ---> JNK">
78            <synonyms>.&#xA;        </synonyms>
79          </chains>
80          <chains id="CHOOOOOO572" name="LPS ---> p38">
81            <synonyms>.&#xA;        </synonyms>
82          </chains>
83          <chains id="CHOOOOOO831" name="LPS ---> IRF-3:IRF-7:CBP:p300">
84            <synonyms>.&#xA;        </synonyms>
85          </chains>
86        </pathway>
87      </transpath:Network>
```

Listado E.1: Instancia en XMI del pathway del TLR4 convertido de forma automática

Apéndice F

Representación en XMI de la red de *Petri* resultante de la transformación

```
<?xml version="1.0" encoding="ASCII"?>
<cpn:Cpnet xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www
    .w3.org/2001/XMLSchema-instance" xmlns:cpn="http:///es.upv.dsic.issi/moment/
    intergenomics/cpn">
  <globbox id="Declarations">
    <declarations xsi:type="cpn:Block" idname="Resources" id="Resources">
      <declarations xsi:type="cpn:Enumerated" idname="O" id="O" usedIn="//@globbox/
          @declarations.1/@declarations.1 //@globbox/@declarations.1/@declarations.0"
          >
        <colorElements name="LPS"/>
      </declarations>
      <declarations xsi:type="cpn:Enumerated" idname="A" id="A" usedIn="//@globbox/
          @declarations.1/@declarations.2 //@globbox/@declarations.1/@declarations.1
          //@globbox/@declarations.1/@declarations.0">
        <colorElements name="TIRAP"/>
        <colorElements name="MyD88"/>
        <colorElements name="LBP"/>
      </declarations>
      <declarations xsi:type="cpn:Enumerated" idname="R" id="R" usedIn="//@globbox/
          @declarations.1/@declarations.2 //@globbox/@declarations.1/@declarations.1"
          >
        <colorElements name="ST2"/>
        <colorElements name="CD14"/>
      </declarations>
    </declarations>
    <declarations xsi:type="cpn:Block" idname="Complexes" id="Complexes">
      <declarations xsi:type="cpn:Product" idname="AO" simpleColors="//@globbox/
          @declarations.0/@declarations.1 //@globbox/@declarations.0/@declarations.0"
          />
      <declarations xsi:type="cpn:Product" idname="AOR" simpleColors="//@globbox/
          @declarations.0/@declarations.1 //@globbox/@declarations.0/@declarations.0
          //@globbox/@declarations.0/@declarations.2"/>
```

```
21            <declarations xsi:type="cpn:Product" idname="AR" simpleColors="//@globbox/
                  @declarations.0/@declarations.1 //@globbox/@declarations.0/@declarations.2"
                  />
22          </declarations>
23        </globbox>
24        <page name="TLR4 pathway" id="CH000000755" posx="200" posy="100" width="1000"
              height="800">
25          <places lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" type="//@globbox/@declarations.0/
                @declarations.0" id="0" arcs="//@page/@arcs.0">
26            <initmark lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" id="0">
27              <marks value="1" colorSetElement="//@globbox/@declarations.0/@declarations.0/
                    @colorElements.0"/>
28            </initmark>
29          </places>
30          <places lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" type="//@globbox/@declarations.1/
                @declarations.0" id="A0" text="" arcs="//@page/@arcs.1 //@page/@arcs.10"/>
31          <places lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" type="//@globbox/@declarations.0/
                @declarations.2" id="R" arcs="//@page/@arcs.2 //@page/@arcs.8 //@page/@arcs.9
                ">
32            <initmark lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" id="R">
33              <marks value="1" colorSetElement="//@globbox/@declarations.0/@declarations.2/
                    @colorElements.1"/>
34              <marks value="2" colorSetElement="//@globbox/@declarations.0/@declarations.2/
                    @colorElements.0"/>
35            </initmark>
36          </places>
37          <places lineColour="Maroon" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" type="//@globbox/@declarations.1/
                @declarations.1" id="LPS:LBP:CD14" text="Dead end" arcs="//@page/@arcs.3"/>
38          <places lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" type="//@globbox/@declarations.0/
                @declarations.1" id="A" arcs="//@page/@arcs.4 //@page/@arcs.6 //@page/@arcs
                .11">
39            <initmark lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" id="A">
40              <marks value="1" colorSetElement="//@globbox/@declarations.0/@declarations.1/
                    @colorElements.0"/>
41              <marks value="1" colorSetElement="//@globbox/@declarations.0/@declarations.1/
                    @colorElements.1"/>
42              <marks value="1" colorSetElement="//@globbox/@declarations.0/@declarations.1/
                    @colorElements.2"/>
43            </initmark>
44          </places>
45          <places lineColour="Maroon" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" type="//@globbox/@declarations.1/
                @declarations.2" id="ST2:TIRAP" text="Dead end" arcs="//@page/@arcs.5"/>
46          <places lineColour="Maroon" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" type="//@globbox/@declarations.1/
                @declarations.2" id="ST2:MyD88" text="Dead end" arcs="//@page/@arcs.7"/>
```

```
47          <transs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" id="LPS + LBP &lt;=> LPS:LBP" text=
                "bind LPS,LBP" arcs="//@page/@arcs.0 //@page/@arcs.1 //@page/@arcs.11"/>
48          <transs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" id="LPS:LBP + CD14 &lt;=>
                LPS:LBP:CD14" text="bind CD14,LPS:LBP" arcs="//@page/@arcs.2 //@page/@arcs.3
                //@page/@arcs.10"/>
49          <transs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" id="ST2 + TIRAP &lt;=> ST2:TIRAP"
                text="bind TIRAP,ST2" arcs="//@page/@arcs.4 //@page/@arcs.5 //@page/@arcs.9"/
                >
50          <transs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" hight="40" width="60" id="ST2 + MyD88 &lt;=> ST2:MyD88"
                text="bind MyD88,ST2" arcs="//@page/@arcs.6 //@page/@arcs.7 //@page/@arcs.8"/
                >
51          <arcs lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.0" currentcyckle="2"
                trans="//@page/@transs.0" id="{LPS} => {LPS + LBP &lt;=> LPS:LBP}">
52            <annot lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="1‘LPS"/>
53          </arcs>
54          <arcs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.1" orientation="TtoP"
                 currentcyckle="2" trans="//@page/@transs.0" id="{LPS + LBP &lt;=> LPS:LBP}
                => {LPS:LBP}">
55            <annot lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="(LBP,LPS)"/>
56          </arcs>
57          <arcs lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.2" currentcyckle="2"
                trans="//@page/@transs.1" id="{CD14} => {LPS:LBP + CD14 &lt;=> LPS:LBP:CD14}"
                >
58            <annot lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="1‘CD14"/>
59          </arcs>
60          <arcs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.3" orientation="TtoP"
                 currentcyckle="2" trans="//@page/@transs.1" id="{LPS:LBP + CD14 &lt;=>
                LPS:LBP:CD14} => {LPS:LBP:CD14}">
61            <annot lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="(LBP,LPS,CD14)"/>
62          </arcs>
63          <arcs lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.4" currentcyckle="2"
                trans="//@page/@transs.2" id="{TIRAP} => {ST2 + TIRAP &lt;=> ST2:TIRAP}">
64            <annot lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="1‘TIRAP"/>
65          </arcs>
66          <arcs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.5" orientation="TtoP"
                 currentcyckle="2" trans="//@page/@transs.2" id="{ST2 + TIRAP &lt;=>
                ST2:TIRAP} => {ST2:TIRAP}">
67            <annot lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="(TIRAP,ST2)"/>
```

```
68          </arcs>
69          <arcs lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.4" currentcyckle="2"
                trans="//@page/@transs.3" id="{MyD88} => {ST2 + MyD88 &lt;=> ST2:MyD88}">
70            <annot lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="1‘MyD88"/>
71          </arcs>
72          <arcs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.6" orientation="TtoP"
                 currentcyckle="2" trans="//@page/@transs.3" id="{ST2 + MyD88 &lt;=>
                ST2:MyD88} => {ST2:MyD88}">
73            <annot lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="(MyD88,ST2)"/>
74          </arcs>
75          <arcs lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.2" currentcyckle="2"
                trans="//@page/@transs.3" id="{ST2} => {ST2 + MyD88 &lt;=> ST2:MyD88}">
76            <annot lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="1‘ST2"/>
77          </arcs>
78          <arcs lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.2" currentcyckle="2"
                trans="//@page/@transs.2" id="{ST2} => {ST2 + TIRAP &lt;=> ST2:TIRAP}">
79            <annot lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="1‘ST2"/>
80          </arcs>
81          <arcs lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.1" currentcyckle="2"
                trans="//@page/@transs.1" id="{LPS:LBP} => {LPS:LBP + CD14 &lt;=>
                LPS:LBP:CD14}">
82            <annot lineColour="Black" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="(LBP,LPS)"/>
83          </arcs>
84          <arcs lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                fillPattern="Solid" headsize="1" place="//@page/@places.4" currentcyckle="2"
                trans="//@page/@transs.0" id="{LBP} => {LPS + LBP &lt;=> LPS:LBP}">
85            <annot lineColour="Lime" lineThick="1" lineType="Solid" fillColour="White"
                  fillPattern="Solid" text="1‘LBP"/>
86          </arcs>
87        </page>
88      </cpn:Cpnet>
```

Listado F.1: Modelo de *CPNTools* obtenido como resultado de la transformación

Apéndice G

Modelo de trazas resultante de la transformación Transpath2Cpn

```
<?xml version="1.0" encoding="ASCII"?>
<traces:TraceabilityModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:traces="http:///es.upv.dsic.issi/traces" name="Transformation">
  <links manipulationRule="NetworkToCpnet">
    <domain href="example.tp#//@reactions.0"/>
    <domain href="example.tp#//@molecules.0"/>
    <domain href="example.tp#/"/>
    <domain href="example.tp#//@reactions.1"/>
    <domain href="example.tp#//@reactions.2"/>
    <domain href="example.tp#//@reactions.3"/>
    <domain href="example.tp#//@molecules.1"/>
    <domain href="example.tp#//@molecules.2"/>
    <domain href="example.tp#//@molecules.3"/>
    <domain href="example.tp#//@molecules.4"/>
    <domain href="example.tp#//@molecules.5"/>
    <domain href="example.tp#//@molecules.6"/>
    <domain href="example.tp#//@molecules.7"/>
    <domain href="example.tp#//@molecules.8"/>
    <domain href="example.tp#//@molecules.9"/>
    <range href="result.cpn#//@page"/>
    <range href="result.cpn#//@globbox"/>
    <range href="result.cpn#/"/>
  </links>
  <links manipulationRule="ReactionToGUIElements">
    <domain href="example.tp#//@reactions.0"/>
    <domain href="example.tp#//@reactions.0/@reactantsCoefficient.0"/>
    <domain href="example.tp#//@reactions.0/@producesCoefficient.0"/>
    <domain href="example.tp#//@reactions.0/@reactantsCoefficient.1"/>
    <range href="result.cpn#//@page/@transs.0"/>
    <range href="result.cpn#//@page"/>
  </links>
  <links manipulationRule="ReactantToPlaceArc">
```

```
32          <domain href="example.tp#//@molecules.0"/>
33          <range href="result.cpn#//@page/@places.0"/>
34          <range href="result.cpn#//@page"/>
35          <range href="result.cpn#//@page/@transs.0"/>
36          <range href="result.cpn#//@page/@arcs.0"/>
37        </links>
38        <links manipulationRule="FillCommonAttributesInPlaces">
39          <range href="result.cpn#//@page/@places.0"/>
40        </links>
41        <links manipulationRule="MoleculeToArcAnnot">
42          <domain href="example.tp#//@molecules.0"/>
43          <range href="result.cpn#//@page/@arcs.0/@annot"/>
44          <range href="result.cpn#//@page/@arcs.0"/>
45        </links>
46        <links manipulationRule="SimpleMoleculeToPlaceType">
47          <domain href="example.tp#//@molecules.0"/>
48          <range href="result.cpn#//@globbox/@declarations.0/@declarations.0"/>
49          <range href="result.cpn#//@page/@places.0"/>
50        </links>
51        <links manipulationRule="SimpleReactantToInitMark">
52          <domain href="example.tp#//@molecules.0"/>
53          <range href="result.cpn#//@page/@places.0/@initmark"/>
54          <range href="result.cpn#//@globbox/@declarations.0/@declarations.0/@colorElements
                .0"/>
55          <range href="result.cpn#//@page/@places.0/@initmark/@marks.0"/>
56          <range href="result.cpn#//@page/@places.0"/>
57        </links>
58        <links manipulationRule="ProductToPlaceArc">
59          <domain href="example.tp#//@molecules.2"/>
60          <range href="result.cpn#//@page/@places.1"/>
61          <range href="result.cpn#//@page"/>
62          <range href="result.cpn#//@page/@transs.0"/>
63          <range href="result.cpn#//@page/@arcs.1"/>
64        </links>
65        <links manipulationRule="FillCommonAttributesInPlaces">
66          <range href="result.cpn#//@page/@places.1"/>
67        </links>
68        <links manipulationRule="MoleculeToArcAnnot">
69          <domain href="example.tp#//@molecules.2"/>
70          <range href="result.cpn#//@page/@arcs.1/@annot"/>
71          <range href="result.cpn#//@page/@arcs.1"/>
72        </links>
73        <links manipulationRule="ComplexMoleculeToPlaceType">
74          <domain href="example.tp#//@molecules.2"/>
75          <range href="result.cpn#//@globbox/@declarations.1/@declarations.0"/>
76          <range href="result.cpn#//@page/@places.1"/>
77        </links>
78        <links manipulationRule="ReactionToGUIElements">
79          <domain href="example.tp#//@reactions.1"/>
80          <domain href="example.tp#//@reactions.1/@reactantsCoefficient.0"/>
81          <domain href="example.tp#//@reactions.1/@producesCoefficient.0"/>
82          <domain href="example.tp#//@reactions.1/@reactantsCoefficient.1"/>
83          <range href="result.cpn#//@page/@transs.1"/>
84          <range href="result.cpn#//@page"/>
```

```
85          </links>
86          <links manipulationRule="ReactantToPlaceArc">
87            <domain href="example.tp#//@molecules.3"/>
88            <range href="result.cpn#//@page/@places.2"/>
89            <range href="result.cpn#//@page"/>
90            <range href="result.cpn#//@page/@transs.1"/>
91            <range href="result.cpn#//@page/@arcs.2"/>
92          </links>
93          <links manipulationRule="FillCommonAttributesInPlaces">
94            <range href="result.cpn#//@page/@places.2"/>
95          </links>
96          <links manipulationRule="MoleculeToArcAnnot">
97            <domain href="example.tp#//@molecules.3"/>
98            <range href="result.cpn#//@page/@arcs.2/@annot"/>
99            <range href="result.cpn#//@page/@arcs.2"/>
100         </links>
101         <links manipulationRule="SimpleMoleculeToPlaceType">
102           <domain href="example.tp#//@molecules.3"/>
103           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2"/>
104           <range href="result.cpn#//@page/@places.2"/>
105         </links>
106         <links manipulationRule="SimpleReactantToInitMark">
107           <domain href="example.tp#//@molecules.3"/>
108           <range href="result.cpn#//@page/@places.2/@initmark"/>
109           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2/@colorElements
                .1"/>
110           <range href="result.cpn#//@page/@places.2/@initmark/@marks.0"/>
111           <range href="result.cpn#//@page/@places.2"/>
112         </links>
113         <links manipulationRule="ProductToPlaceArc">
114           <domain href="example.tp#//@molecules.4"/>
115           <range href="result.cpn#//@page/@places.3"/>
116           <range href="result.cpn#//@page"/>
117           <range href="result.cpn#//@page/@transs.1"/>
118           <range href="result.cpn#//@page/@arcs.3"/>
119         </links>
120         <links manipulationRule="FillCommonAttributesInPlaces">
121           <range href="result.cpn#//@page/@places.3"/>
122         </links>
123         <links manipulationRule="MoleculeToArcAnnot">
124           <domain href="example.tp#//@molecules.4"/>
125           <range href="result.cpn#//@page/@arcs.3/@annot"/>
126           <range href="result.cpn#//@page/@arcs.3"/>
127         </links>
128         <links manipulationRule="ComplexMoleculeToPlaceType">
129           <domain href="example.tp#//@molecules.4"/>
130           <range href="result.cpn#//@globbox/@declarations.1/@declarations.1"/>
131           <range href="result.cpn#//@page/@places.3"/>
132         </links>
133         <links manipulationRule="ReactionToGUIElements">
134           <domain href="example.tp#//@reactions.2"/>
135           <domain href="example.tp#//@reactions.2/@reactantsCoefficient.0"/>
136           <domain href="example.tp#//@reactions.2/@producesCoefficient.0"/>
137           <domain href="example.tp#//@reactions.2/@reactantsCoefficient.1"/>
```

```
138           <range href="result.cpn#//@page/@transs.2"/>
139           <range href="result.cpn#//@page"/>
140         </links>
141         <links manipulationRule="ReactantToPlaceArc">
142           <domain href="example.tp#//@molecules.5"/>
143           <range href="result.cpn#//@page/@places.4"/>
144           <range href="result.cpn#//@page"/>
145           <range href="result.cpn#//@page/@transs.2"/>
146           <range href="result.cpn#//@page/@arcs.4"/>
147         </links>
148         <links manipulationRule="FillCommonAttributesInPlaces">
149           <range href="result.cpn#//@page/@places.4"/>
150         </links>
151         <links manipulationRule="MoleculeToArcAnnot">
152           <domain href="example.tp#//@molecules.5"/>
153           <range href="result.cpn#//@page/@arcs.4/@annot"/>
154           <range href="result.cpn#//@page/@arcs.4"/>
155         </links>
156         <links manipulationRule="SimpleMoleculeToPlaceType">
157           <domain href="example.tp#//@molecules.5"/>
158           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1"/>
159           <range href="result.cpn#//@page/@places.4"/>
160         </links>
161         <links manipulationRule="SimpleReactantToInitMark">
162           <domain href="example.tp#//@molecules.5"/>
163           <range href="result.cpn#//@page/@places.4/@initmark"/>
164           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1/@colorElements
                .0"/>
165           <range href="result.cpn#//@page/@places.4/@initmark/@marks.0"/>
166           <range href="result.cpn#//@page/@places.4"/>
167         </links>
168         <links manipulationRule="ProductToPlaceArc">
169           <domain href="example.tp#//@molecules.7"/>
170           <range href="result.cpn#//@page/@places.5"/>
171           <range href="result.cpn#//@page"/>
172           <range href="result.cpn#//@page/@transs.2"/>
173           <range href="result.cpn#//@page/@arcs.5"/>
174         </links>
175         <links manipulationRule="FillCommonAttributesInPlaces">
176           <range href="result.cpn#//@page/@places.5"/>
177         </links>
178         <links manipulationRule="MoleculeToArcAnnot">
179           <domain href="example.tp#//@molecules.7"/>
180           <range href="result.cpn#//@page/@arcs.5/@annot"/>
181           <range href="result.cpn#//@page/@arcs.5"/>
182         </links>
183         <links manipulationRule="ComplexMoleculeToPlaceType">
184           <domain href="example.tp#//@molecules.7"/>
185           <range href="result.cpn#//@globbox/@declarations.1/@declarations.2"/>
186           <range href="result.cpn#//@page/@places.5"/>
187         </links>
188         <links manipulationRule="ReactionToGUIElements">
189           <domain href="example.tp#//@reactions.3"/>
190           <domain href="example.tp#//@reactions.3/@reactantsCoefficient.0"/>
```

```
191           <domain href="example.tp#//@reactions.3/@producesCoefficient.0"/>
192           <domain href="example.tp#//@reactions.3/@reactantsCoefficient.1"/>
193           <range href="result.cpn#//@page/@transs.3"/>
194           <range href="result.cpn#//@page"/>
195         </links>
196         <links manipulationRule="ReactantToPlaceArc">
197           <domain href="example.tp#//@molecules.8"/>
198           <range href="result.cpn#//@page/@places.4"/>
199           <range href="result.cpn#//@page"/>
200           <range href="result.cpn#//@page/@transs.3"/>
201           <range href="result.cpn#//@page/@arcs.6"/>
202         </links>
203         <links manipulationRule="MoleculeToArcAnnot">
204           <domain href="example.tp#//@molecules.8"/>
205           <range href="result.cpn#//@page/@arcs.6/@annot"/>
206           <range href="result.cpn#//@page/@arcs.6"/>
207         </links>
208         <links manipulationRule="SimpleMoleculeToPlaceType">
209           <domain href="example.tp#//@molecules.8"/>
210           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1"/>
211           <range href="result.cpn#//@page/@places.4"/>
212         </links>
213         <links manipulationRule="SimpleReactantToInitMark">
214           <domain href="example.tp#//@molecules.8"/>
215           <range href="result.cpn#//@page/@places.4/@initmark"/>
216           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1/@colorElements
                .1"/>
217           <range href="result.cpn#//@page/@places.4/@initmark/@marks.1"/>
218           <range href="result.cpn#//@page/@places.4"/>
219         </links>
220         <links manipulationRule="ProductToPlaceArc">
221           <domain href="example.tp#//@molecules.9"/>
222           <range href="result.cpn#//@page/@places.6"/>
223           <range href="result.cpn#//@page"/>
224           <range href="result.cpn#//@page/@transs.3"/>
225           <range href="result.cpn#//@page/@arcs.7"/>
226         </links>
227         <links manipulationRule="FillCommonAttributesInPlaces">
228           <range href="result.cpn#//@page/@places.6"/>
229         </links>
230         <links manipulationRule="MoleculeToArcAnnot">
231           <domain href="example.tp#//@molecules.9"/>
232           <range href="result.cpn#//@page/@arcs.7/@annot"/>
233           <range href="result.cpn#//@page/@arcs.7"/>
234         </links>
235         <links manipulationRule="ComplexMoleculeToPlaceType">
236           <domain href="example.tp#//@molecules.9"/>
237           <range href="result.cpn#//@globbox/@declarations.1/@declarations.2"/>
238           <range href="result.cpn#//@page/@places.6"/>
239         </links>
240         <links manipulationRule="ComplexMoleculeToProduct">
241           <domain href="example.tp#//@molecules.0"/>
242           <domain href="example.tp#//@molecules.2"/>
243           <domain href="example.tp#//@molecules.1"/>
```

```
244           <range href="result.cpn#//@globbox/@declarations.1/@declarations.0"/>
245           <range href="result.cpn#//@globbox/@declarations.0"/>
246           <range href="result.cpn#//@globbox"/>
247           <range href="result.cpn#//@globbox/@declarations.1"/>
248         </links>
249         <links manipulationRule="SimpleMoleculeToEnumerated">
250           <domain href="example.tp#//@molecules.0"/>
251           <range href="result.cpn#//@globbox/@declarations.1/@declarations.0"/>
252           <range href="result.cpn#//@globbox/@declarations.0/@declarations.0/@colorElements
                  .0"/>
253           <range href="result.cpn#//@globbox/@declarations.0/@declarations.0"/>
254           <range href="result.cpn#//@globbox/@declarations.0"/>
255         </links>
256         <links manipulationRule="ComplexMoleculeToProduct">
257           <domain href="example.tp#//@molecules.0"/>
258           <domain href="example.tp#//@molecules.4"/>
259           <domain href="example.tp#//@molecules.3"/>
260           <domain href="example.tp#//@molecules.1"/>
261           <range href="result.cpn#//@globbox/@declarations.1/@declarations.1"/>
262           <range href="result.cpn#//@globbox/@declarations.0"/>
263           <range href="result.cpn#//@globbox"/>
264           <range href="result.cpn#//@globbox/@declarations.1"/>
265         </links>
266         <links manipulationRule="SimpleMoleculeToEnumerated">
267           <domain href="example.tp#//@molecules.0"/>
268           <range href="result.cpn#//@globbox/@declarations.1/@declarations.1"/>
269           <range href="result.cpn#//@globbox/@declarations.0/@declarations.0/@colorElements
                  .0"/>
270           <range href="result.cpn#//@globbox/@declarations.0/@declarations.0"/>
271           <range href="result.cpn#//@globbox/@declarations.0"/>
272         </links>
273         <links manipulationRule="ComplexMoleculeToProduct">
274           <domain href="example.tp#//@molecules.5"/>
275           <domain href="example.tp#//@molecules.7"/>
276           <domain href="example.tp#//@molecules.6"/>
277           <range href="result.cpn#//@globbox/@declarations.1/@declarations.2"/>
278           <range href="result.cpn#//@globbox/@declarations.0"/>
279           <range href="result.cpn#//@globbox"/>
280           <range href="result.cpn#//@globbox/@declarations.1"/>
281         </links>
282         <links manipulationRule="SimpleMoleculeToEnumerated">
283           <domain href="example.tp#//@molecules.5"/>
284           <range href="result.cpn#//@globbox/@declarations.1/@declarations.2"/>
285           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1/@colorElements
                  .0"/>
286           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1"/>
287           <range href="result.cpn#//@globbox/@declarations.0"/>
288         </links>
289         <links manipulationRule="ComplexMoleculeToProduct">
290           <domain href="example.tp#//@molecules.8"/>
291           <domain href="example.tp#//@molecules.9"/>
292           <domain href="example.tp#//@molecules.6"/>
293           <range href="result.cpn#//@globbox/@declarations.1/@declarations.2"/>
294           <range href="result.cpn#//@globbox/@declarations.0"/>
```

```
295           <range href="result.cpn#//@globbox"/>
296           <range href="result.cpn#//@globbox/@declarations.1"/>
297         </links>
298         <links manipulationRule="SimpleMoleculeToEnumerated">
299           <domain href="example.tp#//@molecules.8"/>
300           <range href="result.cpn#//@globbox/@declarations.1/@declarations.2"/>
301           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1/@colorElements
                  .1"/>
302           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1"/>
303           <range href="result.cpn#//@globbox/@declarations.0"/>
304         </links>
305         <links manipulationRule="SimpleMoleculeToEnumerated">
306           <domain href="example.tp#//@molecules.6"/>
307           <range href="result.cpn#//@globbox/@declarations.1/@declarations.2"/>
308           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2/@colorElements
                  .0"/>
309           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2"/>
310           <range href="result.cpn#//@globbox/@declarations.0"/>
311         </links>
312         <links manipulationRule="SimpleMoleculeToEnumerated">
313           <domain href="example.tp#//@molecules.3"/>
314           <range href="result.cpn#//@globbox/@declarations.1/@declarations.1"/>
315           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2/@colorElements
                  .1"/>
316           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2"/>
317           <range href="result.cpn#//@globbox/@declarations.0"/>
318         </links>
319         <links manipulationRule="SimpleMoleculeToEnumerated">
320           <domain href="example.tp#//@molecules.1"/>
321           <range href="result.cpn#//@globbox/@declarations.1/@declarations.1"/>
322           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1/@colorElements
                  .2"/>
323           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1"/>
324           <range href="result.cpn#//@globbox/@declarations.0"/>
325         </links>
326         <links manipulationRule="SimpleMoleculeToEnumerated">
327           <domain href="example.tp#//@molecules.1"/>
328           <range href="result.cpn#//@globbox/@declarations.1/@declarations.0"/>
329           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1/@colorElements
                  .2"/>
330           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1"/>
331           <range href="result.cpn#//@globbox/@declarations.0"/>
332         </links>
333         <links manipulationRule="ReactantToPlaceArc">
334           <domain href="example.tp#//@molecules.6"/>
335           <range href="result.cpn#//@page/@places.2"/>
336           <range href="result.cpn#//@page"/>
337           <range href="result.cpn#//@page/@transs.3"/>
338           <range href="result.cpn#//@page/@arcs.8"/>
339         </links>
340         <links manipulationRule="MoleculeToArcAnnot">
341           <domain href="example.tp#//@molecules.6"/>
342           <range href="result.cpn#//@page/@arcs.8/@annot"/>
343           <range href="result.cpn#//@page/@arcs.8"/>
```

```
344         </links>
345         <links manipulationRule="SimpleMoleculeToPlaceType">
346           <domain href="example.tp#//@molecules.6"/>
347           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2"/>
348           <range href="result.cpn#//@page/@places.2"/>
349         </links>
350         <links manipulationRule="SimpleReactantToInitMark">
351           <domain href="example.tp#//@molecules.6"/>
352           <range href="result.cpn#//@page/@places.2/@initmark"/>
353           <range href="result.cpn#//@globbox/@declarations.0/@declarations.2/@colorElements
                  .0"/>
354           <range href="result.cpn#//@page/@places.2/@initmark/@marks.1"/>
355           <range href="result.cpn#//@page/@places.2"/>
356         </links>
357         <links manipulationRule="ReactantToPlaceArc">
358           <domain href="example.tp#//@molecules.6"/>
359           <range href="result.cpn#//@page/@places.2"/>
360           <range href="result.cpn#//@page"/>
361           <range href="result.cpn#//@page/@transs.2"/>
362           <range href="result.cpn#//@page/@arcs.9"/>
363         </links>
364         <links manipulationRule="MoleculeToArcAnnot">
365           <domain href="example.tp#//@molecules.6"/>
366           <range href="result.cpn#//@page/@arcs.9/@annot"/>
367           <range href="result.cpn#//@page/@arcs.9"/>
368         </links>
369         <links manipulationRule="ReactantToPlaceArc">
370           <domain href="example.tp#//@molecules.2"/>
371           <range href="result.cpn#//@page/@places.1"/>
372           <range href="result.cpn#//@page"/>
373           <range href="result.cpn#//@page/@transs.1"/>
374           <range href="result.cpn#//@page/@arcs.10"/>
375         </links>
376         <links manipulationRule="MoleculeToArcAnnot">
377           <domain href="example.tp#//@molecules.2"/>
378           <range href="result.cpn#//@page/@arcs.10/@annot"/>
379           <range href="result.cpn#//@page/@arcs.10"/>
380         </links>
381         <links manipulationRule="ReactantToPlaceArc">
382           <domain href="example.tp#//@molecules.1"/>
383           <range href="result.cpn#//@page/@places.4"/>
384           <range href="result.cpn#//@page"/>
385           <range href="result.cpn#//@page/@transs.0"/>
386           <range href="result.cpn#//@page/@arcs.11"/>
387         </links>
388         <links manipulationRule="MoleculeToArcAnnot">
389           <domain href="example.tp#//@molecules.1"/>
390           <range href="result.cpn#//@page/@arcs.11/@annot"/>
391           <range href="result.cpn#//@page/@arcs.11"/>
392         </links>
393         <links manipulationRule="SimpleMoleculeToPlaceType">
394           <domain href="example.tp#//@molecules.1"/>
395           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1"/>
396           <range href="result.cpn#//@page/@places.4"/>
```

```
397         </links>
398         <links manipulationRule="SimpleReactantToInitMark">
399           <domain href="example.tp#//@molecules.1"/>
400           <range href="result.cpn#//@page/@places.4/@initmark"/>
401           <range href="result.cpn#//@globbox/@declarations.0/@declarations.1/@colorElements
                  .2"/>
402           <range href="result.cpn#//@page/@places.4/@initmark/@marks.2"/>
403           <range href="result.cpn#//@page/@places.4"/>
404         </links>
405         <domainModels>platform:/resource/Transpath2CpnMedini/example.tp</domainModels>
406         <targetModels>platform:/resource/Transpath2CpnMedini/result.cpn</targetModels>
407       </traces:TraceabilityModel>
```

Listado G.1: Modelo de trazabilidad resultante de la transformación

Apéndice H

Representación en XML de *CPNTools* de la red de *Petri* resultante

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE workspaceElements PUBLIC "-//CPN//DTD CPNXML 1.0//EN"
                                    "http://www.daimi.au.dk/~cpntools/bin/DTD/5/cpn.dtd">
<workspaceElements>
  <generator format="5" tool="CPN Tools" version="2.0.0"/>
  <cpnet>
    <globbox>
      <block id="ID28160336">
        <id>Resources</id>
        <color id="ID5515041">
          <id>O</id>
          <enum>
            <id>LPS</id>
          </enum>
          <layout>colset O = with LPS;</layout>
        </color>
        <color id="ID13969490">
          <id>A</id>
          <enum>
            <id>TIRAP</id>
            <id>MyD88</id>
            <id>LBP</id>
          </enum>
          <layout>colset A = with TIRAP | MyD88 | LBP;</layout>
        </color>
        <color id="ID4867026">
          <id>R</id>
          <enum>
```

```
29                  <id>ST2</id>
30                  <id>CD14</id>
31                </enum>
32                <layout>colset R = with ST2 | CD14;</layout>
33              </color>
34            </block>
35            <block id="ID27966265">
36              <id>Complexes</id>
37              <color id="ID32098342">
38                <id>AO</id>
39                <product>
40                  <id>A</id>
41                  <id>O</id>
42                </product>
43                <layout>colset AO = product A*O;</layout>
44              </color>
45              <color id="ID22232031">
46                <id>AOR</id>
47                <product>
48                  <id>A</id>
49                  <id>O</id>
50                  <id>R</id>
51                </product>
52                <layout>colset AOR = product A*O*R;</layout>
53              </color>
54              <color id="ID25155637">
55                <id>AR</id>
56                <product>
57                  <id>A</id>
58                  <id>R</id>
59                </product>
60                <layout>colset AR = product A*R;</layout>
61              </color>
62            </block>
63          </globbox>
64          <page id="ID4964337">
65            <pageattr name="TLR4 pathway"/>
66            <place id="ID19305075">
67              <posattr x="416.000000" y="-321.000000"/>
68              <fillattr colour="White" filled="false" pattern="Solid"/>
69              <lineattr colour="Lime" thick="1" type="Solid"/>
70              <textattr bold="false" colour="Lime"/>
71              <text/>
72              <ellipse h="40.0" w="60.0"/>
73              <token x="-10.0" y="0.0"/>
74              <marking x="0.0" y="0.0"/>
75              <type id="ID22600021">
76                <posattr x="446.0" y="-341.0"/>
77                <fillattr colour="White" filled="false" pattern="Solid"/>
78                <lineattr colour="Lime" thick="1" type="Solid"/>
79                <textattr bold="false" colour="Lime"/>
80                <text tool="CPN Tools" version="2.0.0">O</text>
81              </type>
82              <initmark id="ID2385247">
```

```
83                <posattr x="476.000000" y="-281.000000"/>
84                <fillattr colour="White" filled="false" pattern="Solid"/>
85                <lineattr colour="Lime" thick="1" type="Solid"/>
86                <textattr bold="false" colour="Lime"/>
87                <text tool="CPN Tools" version="2.0.0">1‘LPS</text>
88              </initmark>
89            </place>
90            <place id="ID24860579">
91              <posattr x="407.000000" y="2.000000"/>
92              <fillattr colour="White" filled="false" pattern="Solid"/>
93              <lineattr colour="Black" thick="1" type="Solid"/>
94              <textattr bold="false" colour="Black"/>
95              <text/>
96              <ellipse h="40.0" w="60.0"/>
97              <token x="-10.0" y="0.0"/>
98              <marking x="0.0" y="0.0"/>
99              <type id="ID11975475">
100               <posattr x="437.0" y="-18.0"/>
101               <fillattr colour="White" filled="false" pattern="Solid"/>
102               <lineattr colour="Black" thick="1" type="Solid"/>
103               <textattr bold="false" colour="Black"/>
104               <text tool="CPN Tools" version="2.0.0">AO</text>
105             </type>
106           </place>
107           <place id="ID12338150">
108             <posattr x="-63.000000" y="116.000000"/>
109             <fillattr colour="White" filled="false" pattern="Solid"/>
110             <lineattr colour="Lime" thick="1" type="Solid"/>
111             <textattr bold="false" colour="Lime"/>
112             <text/>
113             <ellipse h="40.0" w="60.0"/>
114             <token x="-10.0" y="0.0"/>
115             <marking x="0.0" y="0.0"/>
116             <type id="ID907103">
117               <posattr x="-33.0" y="96.0"/>
118               <fillattr colour="White" filled="false" pattern="Solid"/>
119               <lineattr colour="Lime" thick="1" type="Solid"/>
120               <textattr bold="false" colour="Lime"/>
121               <text tool="CPN Tools" version="2.0.0">R</text>
122             </type>
123             <initmark id="ID11969838">
124               <posattr x="-3.000000" y="156.000000"/>
125               <fillattr colour="White" filled="false" pattern="Solid"/>
126               <lineattr colour="Lime" thick="1" type="Solid"/>
127               <textattr bold="false" colour="Lime"/>
128               <text tool="CPN Tools" version="2.0.0">1‘CD14++
129     2‘ST2</text>
130             </initmark>
131           </place>
132           <place id="ID18253559">
133             <posattr x="421.000000" y="317.000000"/>
134             <fillattr colour="White" filled="false" pattern="Solid"/>
135             <lineattr colour="Maroon" thick="1" type="Solid"/>
136             <textattr bold="false" colour="Maroon"/>
```

```
137               <text>Dead end</text>
138               <ellipse h="40.0" w="60.0"/>
139               <token x="-10.0" y="0.0"/>
140               <marking x="0.0" y="0.0"/>
141               <type id="ID32960466">
142                 <posattr x="451.0" y="297.0"/>
143                 <fillattr colour="White" filled="false" pattern="Solid"/>
144                 <lineattr colour="Maroon" thick="1" type="Solid"/>
145                 <textattr bold="false" colour="Maroon"/>
146                 <text tool="CPN Tools" version="2.0.0">AOR</text>
147               </type>
148             </place>
149             <place id="ID16445461">
150               <posattr x="-65.000000" y="-106.000000"/>
151               <fillattr colour="White" filled="false" pattern="Solid"/>
152               <lineattr colour="Lime" thick="1" type="Solid"/>
153               <textattr bold="false" colour="Lime"/>
154               <text/>
155               <ellipse h="40.0" w="60.0"/>
156               <token x="-10.0" y="0.0"/>
157               <marking x="0.0" y="0.0"/>
158               <type id="ID25489408">
159                 <posattr x="-35.0" y="-126.0"/>
160                 <fillattr colour="White" filled="false" pattern="Solid"/>
161                 <lineattr colour="Lime" thick="1" type="Solid"/>
162                 <textattr bold="false" colour="Lime"/>
163                 <text tool="CPN Tools" version="2.0.0">A</text>
164               </type>
165               <initmark id="ID11054201">
166                 <posattr x="-5.000000" y="-66.000000"/>
167                 <fillattr colour="White" filled="false" pattern="Solid"/>
168                 <lineattr colour="Lime" thick="1" type="Solid"/>
169                 <textattr bold="false" colour="Lime"/>
170                 <text tool="CPN Tools" version="2.0.0">1‘TIRAP++
171       1‘MyD88++
172       1‘LBP</text>
173               </initmark>
174             </place>
175             <place id="ID30636498">
176               <posattr x="-411.000000" y="320.000000"/>
177               <fillattr colour="White" filled="false" pattern="Solid"/>
178               <lineattr colour="Maroon" thick="1" type="Solid"/>
179               <textattr bold="false" colour="Maroon"/>
180               <text>Dead end</text>
181               <ellipse h="40.0" w="60.0"/>
182               <token x="-10.0" y="0.0"/>
183               <marking x="0.0" y="0.0"/>
184               <type id="ID14864136">
185                 <posattr x="-381.0" y="300.0"/>
186                 <fillattr colour="White" filled="false" pattern="Solid"/>
187                 <lineattr colour="Maroon" thick="1" type="Solid"/>
188                 <textattr bold="false" colour="Maroon"/>
189                 <text tool="CPN Tools" version="2.0.0">AR</text>
190               </type>
```

```
191             </place>
192             <place id="ID8791042">
193               <posattr x="-423.000000" y="-323.000000"/>
194               <fillattr colour="White" filled="false" pattern="Solid"/>
195               <lineattr colour="Maroon" thick="1" type="Solid"/>
196               <textattr bold="false" colour="Maroon"/>
197               <text>Dead end</text>
198               <ellipse h="40.0" w="60.0"/>
199               <token x="-10.0" y="0.0"/>
200               <marking x="0.0" y="0.0"/>
201               <type id="ID10883068">
202                 <posattr x="-393.0" y="-343.0"/>
203                 <fillattr colour="White" filled="false" pattern="Solid"/>
204                 <lineattr colour="Maroon" thick="1" type="Solid"/>
205                 <textattr bold="false" colour="Maroon"/>
206                 <text tool="CPN Tools" version="2.0.0">AR</text>
207               </type>
208             </place>
209             <trans explicit="false" id="ID31925350">
210               <posattr x="244.000000" y="-195.000000"/>
211               <fillattr colour="White" filled="false" pattern="Solid"/>
212               <lineattr colour="Black" thick="1" type="Solid"/>
213               <textattr bold="false" colour="Black"/>
214               <text>bind
215       LPS,LBP</text>
216               <box h="40.0" w="60.0"/>
217               <binding x="7.2" y="-3.0"/>
218             </trans>
219             <trans explicit="false" id="ID444779">
220               <posattr x="247.000000" y="201.000000"/>
221               <fillattr colour="White" filled="false" pattern="Solid"/>
222               <lineattr colour="Black" thick="1" type="Solid"/>
223               <textattr bold="false" colour="Black"/>
224               <text>bind
225       CD14,LPS:LBP</text>
226               <box h="40.0" w="60.0"/>
227               <binding x="7.2" y="-3.0"/>
228             </trans>
229             <trans explicit="false" id="ID26319903">
230               <posattr x="-259.000000" y="149.000000"/>
231               <fillattr colour="White" filled="false" pattern="Solid"/>
232               <lineattr colour="Black" thick="1" type="Solid"/>
233               <textattr bold="false" colour="Black"/>
234               <text>bind
235       TIRAP,ST2</text>
236               <box h="40.0" w="60.0"/>
237               <binding x="7.2" y="-3.0"/>
238             </trans>
239             <trans explicit="false" id="ID10546001">
240               <posattr x="-263.000000" y="-136.000000"/>
241               <fillattr colour="White" filled="false" pattern="Solid"/>
242               <lineattr colour="Black" thick="1" type="Solid"/>
243               <textattr bold="false" colour="Black"/>
244               <text>bind
```

```
245       MyD88,ST2</text>
246               <box h="40.0" w="60.0"/>
247               <binding x="7.2" y="-3.0"/>
248             </trans>
249             <arc id="ID12136681" order="0" orientation="PtoT">
250               <posattr x="0.000000" y="0.000000"/>
251               <fillattr colour="White" filled="false" pattern="Solid"/>
252               <lineattr colour="Lime" thick="1" type="Solid"/>
253               <textattr bold="false" colour="Lime"/>
254               <arrowattr currentcyckle="2" headsize="1"/>
255               <transend idref="ID31925350"/>
256               <placeend idref="ID19305075"/>
257               <annot id="ID3808947">
258                 <posattr x="330.000000" y="-258.000000"/>
259                 <fillattr colour="White" filled="false" pattern="Solid"/>
260                 <lineattr colour="Lime" thick="1" type="Solid"/>
261                 <textattr bold="false" colour="Lime"/>
262                 <text tool="CPN Tools" version="2.0.0">1‘LPS</text>
263               </annot>
264             </arc>
265             <arc id="ID28685666" order="0" orientation="TtoP">
266               <posattr x="0.000000" y="0.000000"/>
267               <fillattr colour="White" filled="false" pattern="Solid"/>
268               <lineattr colour="Black" thick="1" type="Solid"/>
269               <textattr bold="false" colour="Black"/>
270               <arrowattr currentcyckle="2" headsize="1"/>
271               <transend idref="ID31925350"/>
272               <placeend idref="ID24860579"/>
273               <annot id="ID9912073">
274                 <posattr x="325.000000" y="-97.000000"/>
275                 <fillattr colour="White" filled="false" pattern="Solid"/>
276                 <lineattr colour="Black" thick="1" type="Solid"/>
277                 <textattr bold="false" colour="Black"/>
278                 <text tool="CPN Tools" version="2.0.0">(LBP,LPS)</text>
279               </annot>
280             </arc>
281             <arc id="ID28027084" order="0" orientation="PtoT">
282               <posattr x="0.000000" y="0.000000"/>
283               <fillattr colour="White" filled="false" pattern="Solid"/>
284               <lineattr colour="Lime" thick="1" type="Solid"/>
285               <textattr bold="false" colour="Lime"/>
286               <arrowattr currentcyckle="2" headsize="1"/>
287               <transend idref="ID444779"/>
288               <placeend idref="ID12338150"/>
289               <annot id="ID14100610">
290                 <posattr x="92.000000" y="158.000000"/>
291                 <fillattr colour="White" filled="false" pattern="Solid"/>
292                 <lineattr colour="Lime" thick="1" type="Solid"/>
293                 <textattr bold="false" colour="Lime"/>
294                 <text tool="CPN Tools" version="2.0.0">1‘CD14</text>
295               </annot>
296             </arc>
297             <arc id="ID27562262" order="0" orientation="TtoP">
298               <posattr x="0.000000" y="0.000000"/>
```

```
299               <fillattr colour="White" filled="false" pattern="Solid"/>
300               <lineattr colour="Black" thick="1" type="Solid"/>
301               <textattr bold="false" colour="Black"/>
302               <arrowattr currentcyckle="2" headsize="1"/>
303               <transend idref="ID444779"/>
304               <placeend idref="ID18253559"/>
305               <annot id="ID20011505">
306                 <posattr x="334.000000" y="259.000000"/>
307                 <fillattr colour="White" filled="false" pattern="Solid"/>
308                 <lineattr colour="Black" thick="1" type="Solid"/>
309                 <textattr bold="false" colour="Black"/>
310                 <text tool="CPN Tools" version="2.0.0">(LBP,LPS,CD14)</text>
311               </annot>
312             </arc>
313             <arc id="ID17299058" order="0" orientation="PtoT">
314               <posattr x="0.000000" y="0.000000"/>
315               <fillattr colour="White" filled="false" pattern="Solid"/>
316               <lineattr colour="Lime" thick="1" type="Solid"/>
317               <textattr bold="false" colour="Lime"/>
318               <arrowattr currentcyckle="2" headsize="1"/>
319               <transend idref="ID26319903"/>
320               <placeend idref="ID16445461"/>
321               <annot id="ID29673032">
322                 <posattr x="-162.000000" y="21.000000"/>
323                 <fillattr colour="White" filled="false" pattern="Solid"/>
324                 <lineattr colour="Lime" thick="1" type="Solid"/>
325                 <textattr bold="false" colour="Lime"/>
326                 <text tool="CPN Tools" version="2.0.0">1`TIRAP</text>
327               </annot>
328             </arc>
329             <arc id="ID28689338" order="0" orientation="TtoP">
330               <posattr x="0.000000" y="0.000000"/>
331               <fillattr colour="White" filled="false" pattern="Solid"/>
332               <lineattr colour="Black" thick="1" type="Solid"/>
333               <textattr bold="false" colour="Black"/>
334               <arrowattr currentcyckle="2" headsize="1"/>
335               <transend idref="ID26319903"/>
336               <placeend idref="ID30636498"/>
337               <annot id="ID4521665">
338                 <posattr x="-335.000000" y="234.000000"/>
339                 <fillattr colour="White" filled="false" pattern="Solid"/>
340                 <lineattr colour="Black" thick="1" type="Solid"/>
341                 <textattr bold="false" colour="Black"/>
342                 <text tool="CPN Tools" version="2.0.0">(TIRAP,ST2)</text>
343               </annot>
344             </arc>
345             <arc id="ID28395157" order="0" orientation="PtoT">
346               <posattr x="0.000000" y="0.000000"/>
347               <fillattr colour="White" filled="false" pattern="Solid"/>
348               <lineattr colour="Lime" thick="1" type="Solid"/>
349               <textattr bold="false" colour="Lime"/>
350               <arrowattr currentcyckle="2" headsize="1"/>
351               <transend idref="ID10546001"/>
352               <placeend idref="ID16445461"/>
```

```
353               <annot id="ID26274905">
354                 <posattr x="-164.000000" y="-121.000000"/>
355                 <fillattr colour="White" filled="false" pattern="Solid"/>
356                 <lineattr colour="Lime" thick="1" type="Solid"/>
357                 <textattr bold="false" colour="Lime"/>
358                 <text tool="CPN Tools" version="2.0.0">1`MyD88</text>
359               </annot>
360             </arc>
361             <arc id="ID26018914" order="0" orientation="TtoP">
362               <posattr x="0.000000" y="0.000000"/>
363               <fillattr colour="White" filled="false" pattern="Solid"/>
364               <lineattr colour="Black" thick="1" type="Solid"/>
365               <textattr bold="false" colour="Black"/>
366               <arrowattr currentcyckle="2" headsize="1"/>
367               <transend idref="ID10546001"/>
368               <placeend idref="ID8791042"/>
369               <annot id="ID18539694">
370                 <posattr x="-343.000000" y="-229.000000"/>
371                 <fillattr colour="White" filled="false" pattern="Solid"/>
372                 <lineattr colour="Black" thick="1" type="Solid"/>
373                 <textattr bold="false" colour="Black"/>
374                 <text tool="CPN Tools" version="2.0.0">(MyD88,ST2)</text>
375               </annot>
376             </arc>
377             <arc id="ID10184539" order="0" orientation="PtoT">
378               <posattr x="0.000000" y="0.000000"/>
379               <fillattr colour="White" filled="false" pattern="Solid"/>
380               <lineattr colour="Lime" thick="1" type="Solid"/>
381               <textattr bold="false" colour="Lime"/>
382               <arrowattr currentcyckle="2" headsize="1"/>
383               <transend idref="ID10546001"/>
384               <placeend idref="ID12338150"/>
385               <annot id="ID10402813">
386                 <posattr x="-163.000000" y="-10.000000"/>
387                 <fillattr colour="White" filled="false" pattern="Solid"/>
388                 <lineattr colour="Lime" thick="1" type="Solid"/>
389                 <textattr bold="false" colour="Lime"/>
390                 <text tool="CPN Tools" version="2.0.0">1`ST2</text>
391               </annot>
392             </arc>
393             <arc id="ID21500582" order="0" orientation="PtoT">
394               <posattr x="0.000000" y="0.000000"/>
395               <fillattr colour="White" filled="false" pattern="Solid"/>
396               <lineattr colour="Lime" thick="1" type="Solid"/>
397               <textattr bold="false" colour="Lime"/>
398               <arrowattr currentcyckle="2" headsize="1"/>
399               <transend idref="ID26319903"/>
400               <placeend idref="ID12338150"/>
401               <annot id="ID12618340">
402                 <posattr x="-161.000000" y="132.000000"/>
403                 <fillattr colour="White" filled="false" pattern="Solid"/>
404                 <lineattr colour="Lime" thick="1" type="Solid"/>
405                 <textattr bold="false" colour="Lime"/>
406                 <text tool="CPN Tools" version="2.0.0">1`ST2</text>
```

```
407               </annot>
408             </arc>
409             <arc id="ID12436803" order="0" orientation="PtoT">
410               <posattr x="0.000000" y="0.000000"/>
411               <fillattr colour="White" filled="false" pattern="Solid"/>
412               <lineattr colour="Black" thick="1" type="Solid"/>
413               <textattr bold="false" colour="Black"/>
414               <arrowattr currentcyckle="2" headsize="1"/>
415               <transend idref="ID444779"/>
416               <placeend idref="ID24860579"/>
417               <annot id="ID14807734">
418                 <posattr x="327.000000" y="101.000000"/>
419                 <fillattr colour="White" filled="false" pattern="Solid"/>
420                 <lineattr colour="Black" thick="1" type="Solid"/>
421                 <textattr bold="false" colour="Black"/>
422                 <text tool="CPN Tools" version="2.0.0">(LBP,LPS)</text>
423               </annot>
424             </arc>
425             <arc id="ID375389" order="0" orientation="PtoT">
426               <posattr x="0.000000" y="0.000000"/>
427               <fillattr colour="White" filled="false" pattern="Solid"/>
428               <lineattr colour="Lime" thick="1" type="Solid"/>
429               <textattr bold="false" colour="Lime"/>
430               <arrowattr currentcyckle="2" headsize="1"/>
431               <transend idref="ID31925350"/>
432               <placeend idref="ID16445461"/>
433               <annot id="ID14795461">
434                 <posattr x="89.000000" y="-150.000000"/>
435                 <fillattr colour="White" filled="false" pattern="Solid"/>
436                 <lineattr colour="Lime" thick="1" type="Solid"/>
437                 <textattr bold="false" colour="Lime"/>
438                 <text tool="CPN Tools" version="2.0.0">1‘LBP</text>
439               </annot>
440             </arc>
441           </page>
442           <instances>
443             <instance id="ID28974123" page="ID4964337"/>
444           </instances>
445           <binders>
446             <cpnbinder height="800.0" id="ID26179333" width="1000.0" x="200.0" y="100.0">
447               <sheets>
448                 <cpnsheet id="ID30343329" instance="ID28974123" zoom="1.0"/>
449               </sheets>
450             </cpnbinder>
451           </binders>
452         </cpnet>
453       </workspaceElements>
```

Listado H.1: Representación en XML de *CPNTools* de la red de *Petri* resultante convertido de forma automática

Apéndice I

Conversor de datos de TRANSPATH en XML a XMI

```
1   package es.upv.dsic.issi.moment.intergenomics.transpath.parser;
2
3   import java.io.File;
4   import java.io.IOException;
5   import java.util.ArrayList;
6   import java.util.List;
7
8   import org.apache.log4j.ConsoleAppender;
9   import org.apache.log4j.FileAppender;
10  import org.apache.log4j.Level;
11  import org.apache.log4j.Logger;
12  import org.apache.log4j.SimpleLayout;
13  import org.eclipse.core.resources.IFile;
14  import org.eclipse.core.runtime.CoreException;
15  import org.eclipse.core.runtime.IPath;
16  import org.eclipse.ui.plugin.AbstractUIPlugin;
17  import org.jdom.Document;
18  import org.jdom.Element;
19  import org.jdom.JDOMException;
20  import org.jdom.input.SAXBuilder;
21  import org.osgi.framework.BundleContext;
22
23  import transpath.Chain;
24  import transpath.EnzymesCoefficient;
25  import transpath.InhibitorsCoefficient;
26  import transpath.Molecule;
27  import transpath.MoleculeType;
28  import transpath.Network;
29  import transpath.Pathway;
30  import transpath.ProductsCoefficient;
31  import transpath.ReactantsCoefficient;
32  import transpath.Reaction;
```

```
33      import transpath.ReactionEffects;
34      import transpath.ReactionType;
35      import transpath.TranspathFactory;
36
37      /**
38       * The activator class controls the plug-in life cycle
39       */
40      public class TranspathParser extends AbstractUIPlugin {
41
42       //...
43
44       static Logger logger = Logger.getLogger("transpath.Pathway");
45       ArrayList<Chain> chains = new ArrayList<Chain>();
46       ArrayList<Molecule> molecules = new ArrayList<Molecule>();
47       ArrayList<Reaction> reactions = new ArrayList<Reaction>();
48
49       //...
50
51       public Network createNetwork(IFile file) throws InitialiseException, CoreException
52       {
53
54        logger.setLevel(Level.DEBUG); //DEBUG < INFO < WARN < ERROR < FATAL
55        Pathway pathway = TranspathFactory.eINSTANCE.createPathway();
56        Network network = TranspathFactory.eINSTANCE.createNetwork();
57        molecules.clear();
58        reactions.clear();
59        chains.clear();
60        network.getPathway().add(pathway);
61
62        try
63        {
64             SimpleLayout layout = new SimpleLayout();
65             ConsoleAppender consoleAppender = new ConsoleAppender( layout );
66             logger.addAppender( consoleAppender );
67             FileAppender fileAppender = new FileAppender( layout, "MeineLogDatei.log",
                   false );
68             logger.addAppender( fileAppender );
69           } catch( Exception ex ) {
70             System.out.println( ex );
71           }
72
73        try
74        {
75         SAXBuilder builder = new SAXBuilder();
76         Document doc = null;
77         doc = builder.build(file.getContents());
78         Element root = doc.getRootElement();
79         boolean onlyOnePathway = true;
80
81         //Element pathwayElement = null;
82         List chainList    = root.getChildren("Pathway");
83         List reactionList = root.getChildren("Reaction");
84         List moleculeList = root.getChildren("Molecule");
85
```

```
86          for (Object clo: chainList)
87          {
88           Element e = (Element) clo;
89           if(e.getChildText("type").equals("pathway"))
90           {
91            if (onlyOnePathway)
92            {
93             //pathwayElement = e;
94             if (e.getChildText("description") != null)
95              pathway.setDescription(e.getChildText("description")); //If the element
                     description doesn't exist the value is null.
96             pathway.setName(e.getChildText("name"));
97             pathway.setId(e.getAttributeValue("id"));
98             if (e.getChildText("synonyms") != null)
99              pathway.getSynonyms().add(e.getChildText("synonyms"));
100            logger.info("Creating pathway object with name " + pathway.getName() + " and id
                     " + pathway.getId());
101            onlyOnePathway = false;
102           }
103           else
104           {
105            logger.warn("In the xml-file are more than one pathway. Only the first one is
                    considered.");
106           }
107          }
108          else if (!e.getChildText("type").equals("chain"))
109           logger.warn("Found a pathway object (ID: " + e.getAttributeValue("id") + " with
                    type unlike pathway or chain. Object will be ignored.");
110         }
111         if (pathway.getId().equals("")  || chainList.isEmpty() || reactionList.isEmpty()
                || moleculeList.isEmpty())
112         {
113          logger.error("The informations in the xml-file are not complete.");
114          throw new InitialiseException("The informations in the xml-file are not complete.
                ");
115         }
116
117         //Create Java-Objects
118         for (Object clo: chainList)
119         {
120          Element e = (Element) clo;
121          if (e.getChildText("type").equals("chain"))
122          {
123           Chain currentChain = TranspathFactory.eINSTANCE.createChain();
124           if (e.getChildText("description") != null)
125            currentChain.setDescription(e.getChildText("description")); //If the element
                     description doesn't exist the value is null.
126           currentChain.setName(e.getChildText("name"));
127           currentChain.setId(e.getAttributeValue("id"));
128           if (e.getChildText("synonyms") != null)
129            currentChain.getSynonyms().add(e.getChildText("synonyms"));
130           if (e.getChildText("pathways") != null)
131           {
132            if (e.getChild("pathways").getChildren("PathwayReference").isEmpty())
```

```
133             {
134              currentChain.setPathway(pathway);
135              pathway.getChains().add(currentChain);
136             }
137             else
138             {
139              for (Object o: e.getChild("pathways").getChildren("PathwayReference"))
140              {
141               if (((Element)o).getAttributeValue("id").equals(pathway.getId()))
142               {
143                currentChain.setPathway(pathway);
144                pathway.getChains().add(currentChain);
145                break;
146               }
147              }
148             }//else
149            }//if (e.getChildText("pathways") != null)
150           }
151          }
152          for (Object rlo: reactionList)
153          {
154           Element e = (Element) rlo;
155           Reaction currentReaction = TranspathFactory.eINSTANCE.createReaction();
156           if (e.getChildText("direct") != null && e.getChildText("direct").equals("direct")
                  )
157           {
158            currentReaction.setDirect(true);
159           }
160           else if (e.getChildText("direct").equals("indirect"))
161           {
162            currentReaction.setDirect(false);
163           }
164           else
165           {
166            currentReaction.setDirect(true);
167            logger.warn("Found a pathway object (ID: " + e.getAttributeValue("id") + " with
                   direct unlike direct or indirect. Value was set to true.");
168           }
169           currentReaction.setId(e.getAttributeValue("id"));
170           currentReaction.setName(e.getChildText("name"));
171           if (e.getChildText("effect") != null)
172            currentReaction.getEffects().add(ReactionEffects.get(e.getChildText("effect")));
173           if (e.getChildText("reversible") != null && e.getChildText("reversible").equals("
                  true"))
174           {
175            currentReaction.setReversible(true);
176           }
177           else if (e.getChildText("reversible") != null && e.getChildText("reversible").
                  equals("false"))
178           {
179            currentReaction.setReversible(false);
180           }
181           else
182           {
```

```
183             currentReaction.setReversible(false);
184             logger.warn("Found a pathway object (ID: " + e.getAttributeValue("id") + " with
                    reversible unlike true or false. Value was set to false.");
185            }
186            if (e.getChildText("type") != null)
187             currentReaction.getReactionType().add(ReactionType.get(e.getChildText("type")));
188            // TODO : This method doesn't exist in the EMF model
189            reactions.add(currentReaction);
190            network.getReactions().add(currentReaction);
191           }//end Reactions
192           for (Object mlo: moleculeList)
193           {
194            Element e = (Element) mlo;
195            Molecule currentMolecule = TranspathFactory.eINSTANCE.createMolecule();
196            currentMolecule.setId(e.getAttributeValue("id"));
197            currentMolecule.setName(e.getChildText("name"));
198            if (e.getChildText("type") != null)
199             currentMolecule.getMoleculeType().add(MoleculeType.get(e.getChildText("type")));
200            if (e.getChildText("klass") != null)
201             currentMolecule.getKlass().add(e.getChildText("klass"));
202            if (e.getChildText("synonyms") != null)
203             currentMolecule.getSynonyms().add(e.getChildText("synonyms"));
204            // TODO : This method doesn't exist in the EMF model
205            molecules.add(currentMolecule);
206            network.getMolecules().add(currentMolecule);
207           }
208
209           //Create association from chain to reaction
210           for (Object clo: chainList)
211           {
212            Element e = (Element) clo;
213            if(e.getChildText("type")!= null && e.getChildText("type").equals("chain"))
214            {
215             Chain currentChain = null;
216             Reaction currentReaction = null;
217             String currentId = "";
218
219             for (Chain c: chains)
220             {
221              if (c.getId().equals(e.getAttributeValue("id")))
222              {
223               currentChain = c;
224               break;
225              }
226             }
227             if (e.getChild("reaction_involved") != null)
228             {
229              for (Object o: e.getChild("reactions_involved").getChildren("ReactionReference"
                     ))
230              {
231               currentReaction = null;
232               currentId = ((Element) o).getAttributeValue("id");
233               for (Reaction r: reactions)
234               {
```

```
235              if (r.getId().equals(currentId))
236              {
237               currentReaction = r;
238               break;
239              }
240             }
241             if (currentReaction != null)
242             {
243              currentChain.getReactionsInvolved().add(currentReaction);
244              currentReaction.getChains().add(currentChain);
245             }
246             else
247             {
248              logger.error("The chain " + currentId + "could not be found in the xml-file."
                     );
249             }
250            }
251           }//if (e.getChild("reaction_involved") != null)
252          }
253         }//for (Element e: chainList)
254
255         //create associations from Reaction to Molecule
256         setReactionMoleculeAssocs(reactionList, "enzymes");
257         setReactionMoleculeAssocs(reactionList, "produces");
258         setReactionMoleculeAssocs(reactionList, "reactants");
259         setReactionMoleculeAssocs(reactionList, "inhibitors");
260
261         //create associations from Molecule to Molecule
262         for (Object oml: moleculeList)
263         {
264          Element e = (Element) oml;
265
266          Molecule m1 = null;
267          Molecule m2 = null;
268          String currentId = "";
269
270          for (Molecule m: molecules)
271          {
272           if (m.getId().equals(e.getAttributeValue("id").trim()))
273           {
274            m1 = m;
275            break;
276           }
277          }
278
279          if (e.getChild("stateofs") != null)
280          {
281           // TODO : stateofs en lugar de states
282           for (Object o: e.getChild("stateofs").getChildren("MoleculeReference"))
283           {
284            m2 = null;
285            currentId = ((Element) o).getAttributeValue("id").trim();
286            for (Molecule m: molecules)
287            {
```

```
288              if (m.getId().equals(currentId))
289              {
290               m2 = m;
291               break;
292              }
293             }
294             if (m2 != null)
295             {
296              // TODO  : This method doesn't exist in the EMF model
297              //m1.addState(m2);
298              m1.getStatesOf().add(m2);
299             }
300             else
301             {
302              logger.warn("The molecule " + currentId + " that is a stateof "+ m1.getId()+"
                     could not be found in the xml-file.");
303             }
304            }
305           }//if (e.getChild("states") != null)
306          }
307         } catch (JDOMException e)
308         {
309          logger.error("Could not create jdom from xml file.");
310          throw new InitialiseException("Could not create jdom from xml file.");
311         } catch (IOException e)
312         {
313          logger.error("Could not read file " + file);
314          throw new InitialiseException("Could not read file " +file);
315         }
316         return network;
317        }//Pathway
318
319        private void setReactionMoleculeAssocs(List reactionList, String molAssoc)
320        {
321
322         for (Object rlo : reactionList)
323         {
324          Element e = (Element) rlo;
325          Reaction currentReaction = null;
326          Molecule currentMolecule = null;
327          String currentId = "";
328          int coef = 0;
329
330          logger.debug("Setting molecule associations "+ molAssoc + " for reaction " + e.
                 getAttributeValue("id") );
331          for (Reaction r : reactions)
332          {
333           if (r.getId().equals(e.getAttributeValue("id").trim()))
334           {
335            currentReaction = r;
336            break;
337           }
338          }
339
```

```
340         if (e.getChild(molAssoc) != null && !e.getChild(molAssoc).getChildren("
                MoleculeReference").isEmpty())
341         {
342          for (Object o : e.getChild(molAssoc).getChildren("MoleculeReference"))
343          {
344
345           currentMolecule = null;
346           currentId = ((Element) o).getAttributeValue("id").trim();
347           System.out.println("currentId:"+currentId);
348           for (Molecule m: molecules)
349           {
350            System.out.println(m.getId());
351            if (m.getId().equals(currentId))
352            {
353             currentMolecule = m;
354             break;
355            }
356           }
357
358           if (currentMolecule != null)
359           {
360            coef = findCoefficient(currentReaction.getName(), currentMolecule.getName());
361            if (coef == -1)
362            {
363             logger.error("Couldn't find the coefficient for the molecule " +
                     currentMolecule.getName() + " in the reation "+ currentReaction.getName()+
                      "\n using -1 as coefficient.");
364            }
365
366            //Add molecule and reaction to the correct lists
367            if (molAssoc.equals("enzymes"))
368            {
369             EnzymesCoefficient coeff = TranspathFactory.eINSTANCE.createEnzymesCoefficient
                     ();
370             coeff.setCoefficient(coef);
371             coeff.setCatalyzes(currentReaction);
372             coeff.setEnzymes(currentMolecule);
373             currentReaction.getEnzymesCoefficient().add(coeff);//adds the molecule
                     automatically to the catalyzesList
374            }
375            else if (molAssoc.equals("produces"))
376            {
377             ProductsCoefficient coeff = TranspathFactory.eINSTANCE.
                     createProductsCoefficient();
378             coeff.setCoefficient(coef);
379             coeff.setRkins(currentReaction);
380             coeff.setProduces(currentMolecule);
381             currentReaction.getProducesCoefficient().add(coeff);
382            }
383            else if (molAssoc.equals("reactants"))
384            {
385             ReactantsCoefficient coeff = TranspathFactory.eINSTANCE.
                     createReactantsCoefficient();
386             coeff.setCoefficient(coef);
```

```
387              coeff.setRkouts(currentReaction);
388              coeff.setReactants(currentMolecule);
389              currentReaction.getReactantsCoefficient().add(coeff);
390             }
391             else if (molAssoc.equals("inhibitors"))
392             {
393              InhibitorsCoefficient coeff = TranspathFactory.eINSTANCE.
                     createInhibitorsCoefficient();
394              coeff.setCoefficient(coef);
395              coeff.setInhibits(currentReaction);
396              coeff.setInhibitors(currentMolecule);
397              currentReaction.getInhibitorsCoefficient().add(coeff);
398             }
399            }//if (currentMolecule != null)
400            else
401            {
402             logger.error("The molecule "+ currentId +" in the reaction "+currentReaction.
                     getId() + " indicated as " + molAssoc + " is missing in the xml-file.");
403            }
404           }//for
405          }//if
406         }//for (Object rlo : reactionList)
407        }//setReactionMoleculeAssocs()
408
409        public int findCoefficient(String reaction, String molecule)
410        {
411         System.out.println("Suche nach "+molecule);
412
413         int coef = 0;
414         int pos = reaction.indexOf(molecule);
415
416         boolean rightSiteOk = false;
417         boolean leftSiteOk  = false;
418         int numberLength = 0;
419
420         while (!leftSiteOk || !rightSiteOk)
421         {
422          rightSiteOk = false;
423          leftSiteOk  = false;
424          numberLength = 0;
425
426          //molecule not found
427          if (pos == -1)
428          {
429           coef = -1;
430           logger.error("No coefficient found, because the molecule " + molecule + " has not
                   been found in the reaction name " + reaction);
431           return coef;
432          }
433          //check right site
434          if ((pos+molecule.length()+1)<reaction.length())
435          {
436           if (reaction.charAt(pos+molecule.length()) == '-' && reaction.charAt(pos+molecule
                   .length()+1) == '>'
```

```
437             || reaction.charAt(pos+molecule.length()) == ',' && reaction.charAt(pos+
                    molecule.length()+1) == ' ')
438           {
439            rightSiteOk = true;
440            System.out.println("rightSiteOk1");
441           }
442          }
443          //check if right site is followed by ' ' or ')' or is the end of the string
444          if ((pos+molecule.length())<reaction.length())
445          {
446           //System.out.println("reaction.charAt(pos+molecule.length())='"+reaction.charAt(
                 pos+molecule.length())+"'");
447           //System.out.println("reaction.charAt(pos+molecule.length()+1)='"+reaction.charAt
                 (pos+molecule.length()+1)+"'");
448           if (reaction.charAt(pos+molecule.length()) == ' ' || ((reaction.charAt(pos+
                 molecule.length()) == ')' && reaction.charAt(pos+molecule.length()+1) == ' ')
                 ))
449           {
450            rightSiteOk = true;
451            System.out.println("rightSiteOk2");
452           }
453          }
454          else
455          {
456           rightSiteOk = true;
457           System.out.println("rightSiteOk3");
458          }
459
460          //check if left site is headed by ' ' or '(' or is the start of the string or is
                 headed by a number
461          //and set coef if the number is the start of the string or headed by ' '
462          if (pos == 0)
463          {
464           System.out.println("leftSiteOk1");
465           leftSiteOk = true;
466           coef = 1;
467          }
468          else if (pos == 1 && reaction.charAt(0) == '(')
469          {
470           System.out.println("leftSiteOk2");
471           leftSiteOk = true;
472           coef = 1;
473          }
474          else if (reaction.charAt(pos-1)== ' ')
475          {
476           System.out.println("leftSiteOk2b");
477           leftSiteOk = true;
478           coef = 1;
479          }
480          else if (pos>1 && reaction.charAt(pos-1) == '(' && reaction.charAt(pos-2) == ' ')
481          {
482           System.out.println("leftSiteOk2c");
483           leftSiteOk = true;
484           coef = 1;
```

```
485          }
486          else if (pos>1 && reaction.charAt(pos-1) == '-' && reaction.charAt(pos-2) == ' ')
487          {
488           coef = 1;
489           System.out.println("leftSiteOk5");
490           leftSiteOk = true;
491          }
492          else
493          {
494           while (reaction.substring(pos-numberLength-1,pos-numberLength).matches("\\d"))
495           {
496
497            System.out.println("'"+reaction.charAt(pos-numberLength-1)+"'");
498            if (pos-numberLength-1 == 0 || reaction.charAt(pos-numberLength-2)== ' '
499             || (pos-numberLength > 2 && reaction.charAt(pos-numberLength-2) == '-'&&
                     reaction.charAt(pos-numberLength-3) == ' '))
500            {
501             System.out.println("leftSiteOk3");
502             leftSiteOk = true;
503             coef = Integer.parseInt(reaction.substring(pos-numberLength-1,pos));
504             System.out.println("coef="+coef);
505             break;
506            }
507            numberLength++;
508           }
509
510           if (pos>1 && reaction.charAt(pos-1) == '(')
511           {
512            while (reaction.substring(pos-numberLength-2,pos-numberLength-1).matches("\\d"))
513            {
514             if (pos-numberLength-2 == 0 || reaction.charAt(pos-numberLength-3)== ' '
515              || (pos-numberLength > 3 && reaction.charAt(pos-numberLength-3) == '-'&&
                       reaction.charAt(pos-numberLength-4) == ' '))
516             {
517              System.out.println("leftSiteOk4");
518              leftSiteOk = true;
519              coef = Integer.parseInt(reaction.substring(pos-numberLength-2,pos-1));
520              break;
521             }
522             numberLength++;
523            }
524           }
525          }
526          pos = reaction.indexOf(molecule,pos+1);
527         }//while
528
529         if (rightSiteOk &&leftSiteOk)
530         {
531          return coef;
532         }
533         else return -1;
534
535        }
```

```
536         }
```

Listado I.1: Conversor de datos de TRANSPATH®en XML a XMI

Apéndice J

Ejecución de una transformación empleando el motor de MediniQVT

```
1  package es.upv.dsic.issi.qvt.launcher.internal;
2
3  import java.io.IOException;
4  import java.io.InputStreamReader;
5  import java.io.Reader;
6  import java.util.ArrayList;
7  import java.util.Collection;
8  import java.util.Collections;
9  import java.util.HashMap;
10
11 import org.eclipse.core.resources.ResourcesPlugin;
12 import org.eclipse.core.resources.WorkspaceJob;
13 import org.eclipse.core.runtime.CoreException;
14 import org.eclipse.core.runtime.IProgressMonitor;
15 import org.eclipse.core.runtime.IStatus;
16 import org.eclipse.core.runtime.Status;
17 import org.eclipse.emf.common.util.Monitor;
18 import org.eclipse.emf.common.util.URI;
19 import org.eclipse.emf.ecore.EObject;
20 import org.eclipse.emf.ecore.EPackage;
21 import org.eclipse.emf.ecore.resource.Resource;
22 import org.eclipse.emf.ecore.resource.ResourceSet;
23 import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
24 import org.eclipse.emf.ecore.xmi.PackageNotFoundException;
25 import org.oslo.ocl20.standard.lib.OclAnyModelElement;
26
27 import traces.TraceabilityLink;
28 import traces.TraceabilityModel;
29 import traces.TracesFactory;
30 import uk.ac.kent.cs.kmf.util.ILog;
31 import uk.ac.kent.cs.kmf.util.OutputStreamLog;
32 import de.ikv.emf.qvt.EMFQvtProcessorImpl;
```

```
33    import de.ikv.medini.qvt.QVTProcessorConsts;
34    import de.ikv.medini.qvt.Trace;
35    import es.upv.dsic.issi.qvt.launcher.QvtLauncherPlugin;
36    import es.upv.dsic.issi.qvt.launcher.model.qvtinvocation.Domain;
37    import es.upv.dsic.issi.qvt.launcher.model.qvtinvocation.QvtTransformationInvocation;
38
39    public class QvtTransformationJob extends WorkspaceJob {
40
41       QvtTransformationInvocation invocation;
42
43       ResourceSet resourceSet = new ResourceSetImpl();
44
45       public QvtTransformationJob(QvtTransformationInvocation invocation) {
46          super("Running " + invocation.getName() + " transformation");
47          this.invocation = invocation;
48       }
49
50       @Override
51       public IStatus runInWorkspace(IProgressMonitor monitor)
52             throws CoreException {
53
54          monitor.beginTask("Running...", Monitor.UNKNOWN);
55
56          Resource targetResource = null;
57
58          ILog log = new OutputStreamLog(System.err);
59
60          EMFQvtProcessorImpl emfQvtProcessorImpl = new EMFQvtProcessorImpl(log);
61
62          Reader qvtScriptReader = new InputStreamReader(
63                ResourcesPlugin.getWorkspace().getRoot().getFile(invocation.getPath()).
                      getContents());
64
65
66          Collection<Resource> models = (new ArrayList<Resource>());
67          for (Domain domain : invocation.getDomains()) {
68
69             Resource resource = null;
70
71             if (ResourcesPlugin.getWorkspace().getRoot().getFile(domain.getModelPath()).
                    exists()) {
72                try {
73                   resource = resourceSet.getResource(
74                         URI.createPlatformResourceURI(domain.getModelPath().toString(),
                                false), true);
75                   resource.load(Collections.EMPTY_MAP);
76
77                   if (!resource.getContents().isEmpty()) {
78                      emfQvtProcessorImpl.addMetaModel(resource.getContents().get(0).
                             eClass().getEPackage());
79                   } else {
80                      resource = resourceSet.createResource(
81                            URI.createPlatformResourceURI(domain.getModelPath().toString
                                  (),false));
```

```
82                      }
83                   } catch (IOException e) {
84                      return new Status(IStatus.ERROR,QvtLauncherPlugin.PLUGIN_ID,e.
                            getLocalizedMessage(),e);
85                   } catch (Exception e) {
86                      if (e.getCause() instanceof PackageNotFoundException) {
87                         return new Status(IStatus.ERROR,QvtLauncherPlugin.PLUGIN_ID,e.
                               getCause().getLocalizedMessage(),e.getCause());
88                      }
89                      return new Status(IStatus.ERROR,QvtLauncherPlugin.PLUGIN_ID,e.
                            getLocalizedMessage(),e);
90                   }
91                } else {
92                   resource = resourceSet.createResource(
93                         URI.createPlatformResourceURI(domain.getModelPath().toString(),
                               false));
94
95                   resource.getContents().clear();
96
97                   Object[] keys = EPackage.Registry.INSTANCE.keySet().toArray();
98
99
100                  for (Object key : keys) {
101                     EPackage pkg = EPackage.Registry.INSTANCE.getEPackage(key.toString());
102                     if (pkg.getNsPrefix().equals(domain.getNsPrefix())) {
103                        emfQvtProcessorImpl.addMetaModel(pkg);
104                     }
105                  }
106
107               }
108               models.add(resource);
109            }
110
111            targetResource = resourceSet.getResource(
112                  URI.createPlatformResourceURI(invocation.getDirection().getModelPath().
                        toString(),false),true);
113
114            targetResource.getContents().clear();
115
116            emfQvtProcessorImpl.setModels(models);
117            emfQvtProcessorImpl.setDebug(true);
118
119
120              emfQvtProcessorImpl.setProperty(QVTProcessorConsts.PROP_DISABLE_TRACES, "true
                      ");
121
122              // Esto hay que deshabilitarlo si se invoca por linea de comandos fuera de
                      eclipse
123              //emfQvtProcessorImpl.setProperty(QVTProcessorConsts.
                      PROP_DISABLE_TRANSACTIONAL_MODE, "true");
124
125              try {
126               Collection<Trace> traces = emfQvtProcessorImpl.evaluateQVT(
127                     qvtScriptReader,
```

```
128                     invocation.getName(),
129                     true,
130                     invocation.getDirection().getName(),
131                     models.toArray(),
132                     new ArrayList<Trace>(),
133                     log);
134             TraceabilityModel traceabilityModel = createTraceabilityModel(traces);
135
136             Resource tracesResource = resourceSet.createResource(
137                   URI.createPlatformResourceURI(invocation.getDirection().getModelPath()
                          .
138                           removeFileExtension().addFileExtension("traces").toString(),
                               false));
139
140             tracesResource.getContents().add(traceabilityModel);
141
142             targetResource.save(Collections.EMPTY_MAP);
143
144             tracesResource.save(Collections.EMPTY_MAP);
145
146           } catch (IOException e) {
147             return new Status(IStatus.ERROR,QvtLauncherPlugin.PLUGIN_ID,e.
                    getLocalizedMessage(),e);
148          } catch (Exception e) {
149             return new Status(IStatus.ERROR,QvtLauncherPlugin.PLUGIN_ID,e.
                    getLocalizedMessage(),e);
150          }
151
152          return Status.OK_STATUS;
153       }
154
155       private TraceabilityModel createTraceabilityModel(Collection<Trace> traces) {
156          TraceabilityModel traceabilityModel = TracesFactory.eINSTANCE.
                createTraceabilityModel();
157
158          traceabilityModel.setName("Transformation");
159
160          for (Domain domain : invocation.getDomains()) {
161             if (domain != invocation.getDirection()) {
162                traceabilityModel.getDomainModels().add(URI.createPlatformResourceURI(
                       domain.getModelPath().toString(),false));
163             } else {
164                traceabilityModel.getTargetModels().add(URI.createPlatformResourceURI(
                       domain.getModelPath().toString(),false));
165             }
166          }
167
168
169          for (Trace t : traces) {
170             TraceabilityLink traceabilityLink = TracesFactory.eINSTANCE.
                    createTraceabilityLink();
171
172             traceabilityModel.getLinks().add(traceabilityLink);
173
```

```
174                traceabilityLink.setManipulationRule(t.getRelation().getName());
175
176                for (Object obj1 : t.getBindings()) {
177                   for (Object obj2 : ((HashMap)obj1).values())
178                   if (obj2 instanceof OclAnyModelElement) {
179                      OclAnyModelElement elt = (OclAnyModelElement) obj2;
180                      if (!((EObject) elt.asJavaObject()).eClass().eResource().equals(
181                            resourceSet.getResource(
182                                  URI.createPlatformResourceURI(invocation.getDirection().
                                        getModelPath().toString(),false),true)
183                                  .getContents().get(0).eClass().eResource())) {
184                         traceabilityLink.getDomain().add((EObject) elt.asJavaObject());
185                      } else {
186                         traceabilityLink.getRange().add((EObject) elt.asJavaObject());
187                      }
188                   }
189                }
190             }
191
192             return traceabilityModel;
193          }
194       }
```

Listado J.1: Implementación del trabajo de ejecución de transformaciones: Clase *QvtTransformationJob*