Estrategias MDE en entornos de desarrollo de sistemas de tiempo real.

César Cuevas, Laura Barros, Patricia López Martínez y José M. Drake
Grupo de Computadores y Tiempo Real
Universidad de Cantabria
39005, Santander, Spain
{cuevasce, barrosl, lopezpa, drakej}@unican.es

Abstract— ***Se describe y analiza la aplicación de la tecnología MDE (Model-Driven Engineering) como soporte de los entornos de diseño de sistemas de tiempo real. Con esta estrategia se incrementa la facilidad de uso del entorno al permitir la definición de modelos orientados a los diferentes paradigmas de diseño de las aplicaciones de tiempo real, y así mismo, se facilita el desarrollo e integración de las herramientas de análisis y diseño de planificabilidad, en base a proporcionar para cada una de ellas un modelo con el nivel de abstracción que necesita. Las herramientas MDE automatizan las transformaciones entre los modelos y el mantenimiento de la coherencia entre ellos.***

Keywords: MDE, Real-time, Modelling

I. INTRODUCCIÓN

Las metodologías y tecnologías MDE [1] (equivalentes a las metodologías MDA y MDD de OMG [2]) constituyen una contribución a la ingeniería software muy relevante porque centran la atención y el razonamiento en los modelos y no en el código como ha sido hasta ahora tradicional. Los modelos abordan la especificación del sistema que se diseña con un mayor nivel de abstracción, lo que facilita su comprensión por un diseñador humano, permiten razonar con independencia de la plataforma de ejecución al ser independientes de los detalles de implementación y son más adecuados para introducir y tratar características no funcionales. Actualmente se aplican con gran éxito en el desarrollo de herramientas de generación de código a partir de modelos, en el desarrollo de tecnologías de componentes y herramientas de ensamblado y despliegue de los mismos y en el desarrollo de pasarelas para la interoperatividad entre entornos de desarrollo de diferentes fabricantes. De forma más limitada y con carácter de prueba, también se ha aplicado la tecnología MDE al diseño de sistemas embebidos y de tiempo real [3][4].

El objetivo de este trabajo es mostrar los desarrollos que se han realizado dentro del proyecto RT-Model [1] para probar y evaluar la tecnología MDE como base de los entornos de análisis, diseño y desarrollo de sistemas de tiempo real.

II. MDE Y DISEÑO DE SISTEMAS DE TIEMPO REAL

El desarrollo de sistemas de tiempo real ha sido siempre un proceso basado en modelos. Tanto los análisis de planificabilidad como las estrategias de diseño y configuración de los parámetros de planificación requieren ser aplicados a través de modelos de comportamiento temporal del sistema que se está desarrollando. Sin embargo, la complejidad actual de los sistemas de tiempo real que se abordan y la sofisticación de las técnicas de análisis y diseño que se aplican, han hecho que se necesite disponer de múltiples modelos complementarios pero sincronizados entre sí. Todo ello conduce a que cada vez sea más costoso integrar herramientas en los entornos.

[1]RT-Model: Plataformas de tiempo real para diseño de sistemas empotrados basado en modelos. Proyecto TIN2008-06766-C03-03

La causa de la diversidad de los modelos se deriva de que en los entornos de diseño de sistemas de tiempo real existen dos vistas complementarias que hay que satisfacer de forma independiente utilizando los modelos adecuados:

1. Desde el punto de vista del desarrollador de aplicaciones de tiempo real, que hace uso del entorno, se requieren modelos en los que los elementos de modelado se correspondan con los elementos que le son familiares: procesadores, redes de comunicación, planificadores, *threads*, operaciones, transacciones *end-to-end-flow*, clases, objetos, componentes, recursos virtuales, contratos de uso de servicio, carga de trabajo, requisitos temporales, etc. Esta vista no es única, sino que su nivel de abstracción cambia según el paradigma de diseño que esté utilizando. Por ejemplo:
    a. En el caso de métodos básicos de desarrollo de sistemas embebidos simples de tiempo real, un modelo de reactividad basado en *threads* y transacciones, operaciones, etc., es el adecuado.
    b. En el caso de que se utilicen paradigmas de diseño modulares como la orientación a objetos o el basado en componentes, los elementos de modelado deben corresponderse a conceptos tales como componentes, objetos (activos, pasivos y protegidos), servicios de middleware, planes de despliegue, etc.
    c. En el caso de que se utilice el paradigma de diseño de reserva de recursos, los modelos utilizan conceptos tales como recursos virtuales, canales de comunicación virtuales, contratos de uso de servicio, plataformas de ejecución virtual, etc.
2. Desde el punto de vista del desarrollador de las herramientas de análisis y diseño de planificabilidad (que constituyen el núcleo fundamental del entorno de desarrollo) se requiere disponer de modelos mucho más simplificados, con sólo los elementos y asociaciones que corresponden a la abstracción que procesan. Por ejemplo:
    a. En el caso de las herramientas de análisis de planificabilidad, la formulación del modelo de acuerdo con un diseño reactivo que describe las transacciones que se ejecutan, el modelo de carga y los requisitos temporales, así como un modelo de recursos que describe los retrasos por suspensión y bloqueo que ocurren como consecuencia de acceso a ellos con exclusión mutua.
    b. En el caso de las herramientas de análisis de planificabilidad basado en reserva de recursos, los niveles de concurrencia, el modelo de carga, los requerimientos temporales, etc.
    c. En el caso de análisis mediante simulación, el modelo debe ser reformulado para que se incremente su eficiencia de ejecución e incorpore los observadores necesarios para que se capture la información de la ejecución que requiere el diseñador.

Como se muestra en la figura 1, el uso de las tecnologías MDE consigue incrementar drásticamente la facilidad de uso del entorno y el desarrollo de las herramientas, en base a que cada diseñador de aplicaciones y cada desarrollador de herramientas puedan disponer del modelo adecuado a su tarea. Las herramientas MDE son las responsables de las transformaciones entre modelos y del mantenimiento de la coherencia entre ellos.

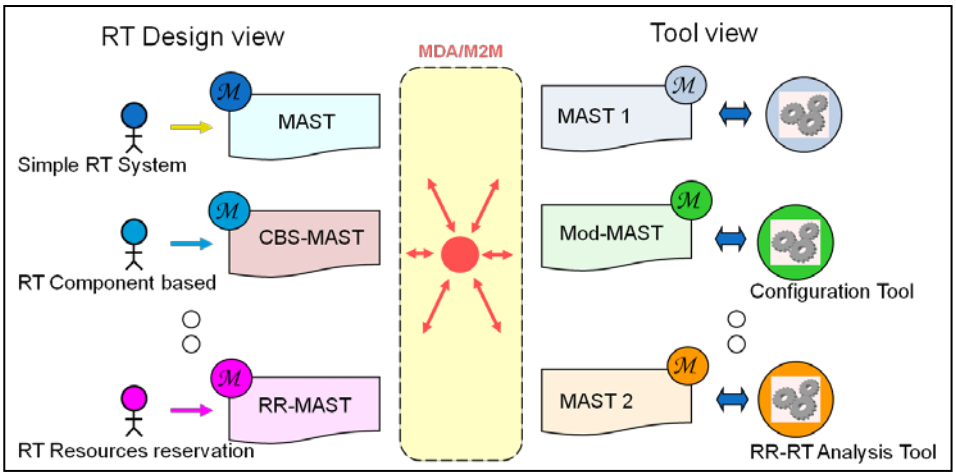

Figura 1: Función de las tecnología MDE en el entorno de desarrollo de sistemas de tiempo real.

En definitiva, las técnicas MDE tienen como objetivos:

1. Transformar los modelos que utilizan los diseñadores de las aplicaciones hasta otros que son adecuados al algoritmo que utiliza la herramienta.
2. Retornar los resultados del análisis a los modelos de diseño, de forma que los diseñadores de las aplicaciones los sepan interpretar.

En el caso del Grupo CTR de la Universidad de Cantabria, un revulsivo muy importante para aplicar esta tecnología ha sido la formulación del entorno de diseño MAST2, que con un único meta-modelo cubre los diferentes modelos que corresponden a los diferentes paradigmas de diseño. Esto simplifica considerablemente las herramientas MDE, ya que en las transformaciones que se requieren, el meta-modelo de partida y destino es MAST2, y por tanto muchos de los elementos de estas herramientas son los mismos.

Como ejemplos, en las secciones III y IV se describen los dos casos que se han desarrollado en el proyecto RT-Model.

III. JSIMMAST

El ejemplo más elaborado que se ha desarrollado ha sido la transformación entre el meta-modelo MAST2 completo (sin restricciones) y el meta-modelo SimMAST, desarrollado para simplificar y hacer más eficiente la ejecución del simulador JSimMAST.

Se cubren con tecnologías MDE todas las fases salvo el propio simulador que es un programa Java:

1. Construcción del modelo a partir del fichero de texto en que se formula.
2. Validación de la coherencia del modelo, lo cual descarga al simulador de cualquier código de verificación.
3. Una transformación muy compleja entre modelos.
4. Gestión de los perfiles de simulación, que introducen los observadores necesarios para que la simulación genere la información que se requiere.
5. Formulación del modelo resultante en el formato que requiere el simulador Java.

La herramienta de simulación tiene todas sus fases integradas en un único comando del entorno Eclipse, y son transparentes para el diseñador de aplicaciones que hace uso de ella.

En la figura 2 se muestra, a modo de ejemplo de la complejidad de la trasformación de modelos MAST2 a SimMAST, la transformación de un elemento Regular_Processor de MAST2 a un conjunto de elementos mucho más simples que reproducen su funcionalidad.

1. El Processor es un simple recurso de exclusión mutua entre *threads*, sin parámetros.
2. La gestión de las interrupciones es realizada por un *thread* en la misma manera que la gestión de la ejecución de código, salvo que es planificado por un planificador que se declara para las interrupciones.
3. Los cambios de contexto por interrupción se gestionan como si fuesen transacciones convencionales de muy alta prioridad.
4. Los tiempos de las operaciones se resuelven a tiempo absoluto en base al procesador en que se llevan a cabo.

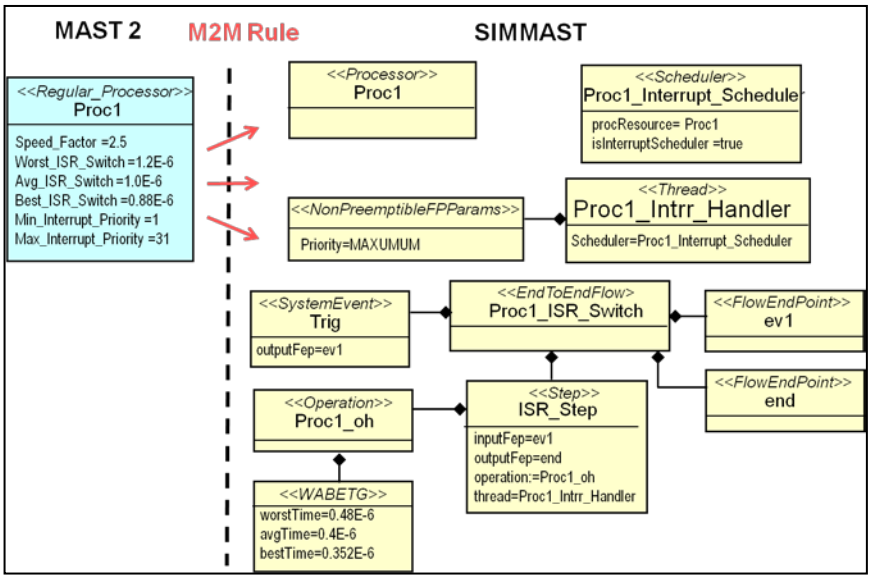

Figura 1: Ejemplo de transformación entre MAST2 y SimMAST.

IV. RESOURCES RESERVATION SUITE

El ejemplo más representativo por su amplitud de la estrategia de uso de MDE es el entorno de desarrollo de aplicaciones de tiempo real basadas en el paradigma de reserva de recursos.

Aquí no hay múltiples meta-modelos, ya que el meta-modelo MAST2 cubre todos los modelos que se requieren para gestionar las diferentes vistas y fases del proceso. Así, las transformaciones entre modelos son siempre endógenas.

1. El diseñador del código puede formular un modelo en el que expresa la reactividad de su funcionalidad.
2. Una herramienta MDE ofrece al diseñador de tiempo real la reactividad, para que éste formule el modelo de carga (número de transacciones y los patrones de disparo), los requisitos temporales, y defina los niveles de concurrencia. De todo ello resulta el *RR_Application_ Model*, que contiene el modelo de la aplicación y el modelo de la plataforma virtual de ejecución.
3. Un conjunto de herramientas permiten al analizador de tiempo real realizar el análisis de tiempo real, esto es, determinar los contratos de uso de los recursos virtuales que establecen los requisitos de disponibilidad de los recursos que se requieren para que la aplicación satisfaga los requisitos temporales con independencia de la plataforma física que en el futuro se despliegue. Dicho conjunto de herramientas permite:

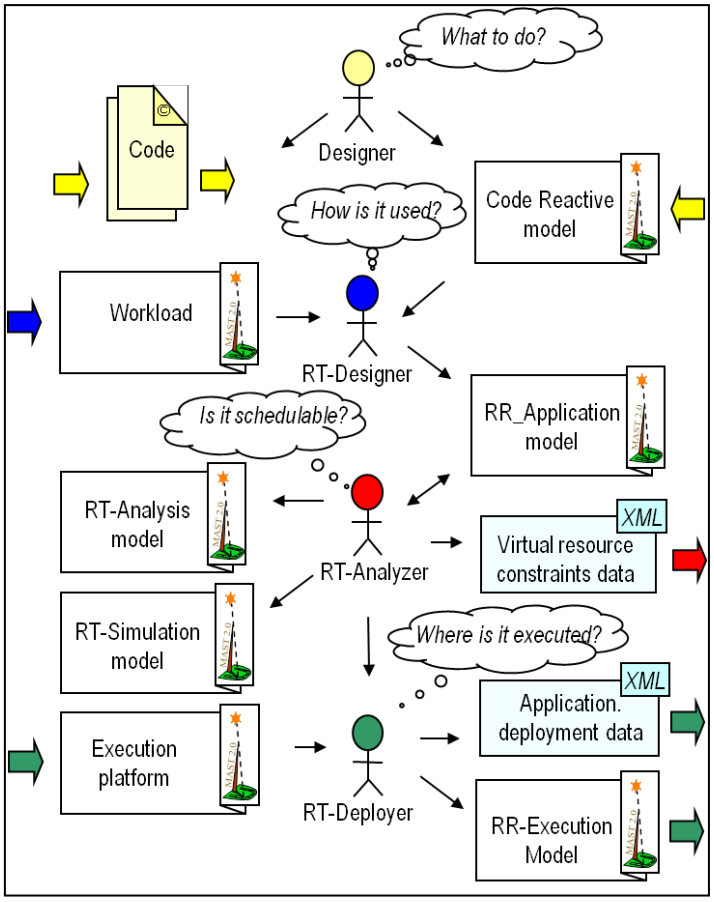

Figura 3: Entorno de sistemas de tiempo real basado en reserva de recursos.

- Procesar el modelo reactivo para estimar los contratos y las restricciones entre sus atributos.
- Generar el modelo que permite, haciendo uso de las herramientas de análisis del entorno MAST, proponer conjuntos de contratos que hacen la aplicación planificable.
- Lo mismo pero haciendo uso del simulador.

4. Una herramienta permite incorporar al *RR_Application_Model* la plataforma física de ejecución y el despliegue sobre ella de la aplicación, y con ello generar el modelo de ejecución que describe cómo ejecutar la aplicación sobre dicha plataforma física.

V. TECNOLOGIA MDE

Según la visión tradicional en tres niveles –modelo (M0) / meta-modelo (M1) / meta-meta-modelo (M2) –, propia del desarrollo de software dirigido por modelos (DSDM) y que se muestra en la figura 4, en la figura 5 se presenta el escenario inicial a partir del cual se han abordado las tareas de transformación de modelos (M2M) que aparecen en el trabajo que aquí se presenta, escenario que viene caracterizado por los requisitos sobre él impuestos. En primer lugar, tanto el meta-modelo origen como el meta-modelo destino, artefactos que conforman la parte nuclear de cualquier transformación M2M, vienen formulados mediante UML2 [5]. Además, se ha impuesto el requerimiento de que tanto el modelo de entrada (conforme al meta-modelo origen) como el modelo de salida (conforme al meta-modelo destino) vengan formulados mediante XML. Este segundo requerimiento implica que, a nivel de meta-modelo, el escenario de partida no se limite a los dos artefactos UML sino que deba expandirse recogiendo la semántica de ambos meta-modelos en sendas plantillas W3C-Schema [6] (*schemas* de origen y de destino) frente a las cuales han de validar los modelos.

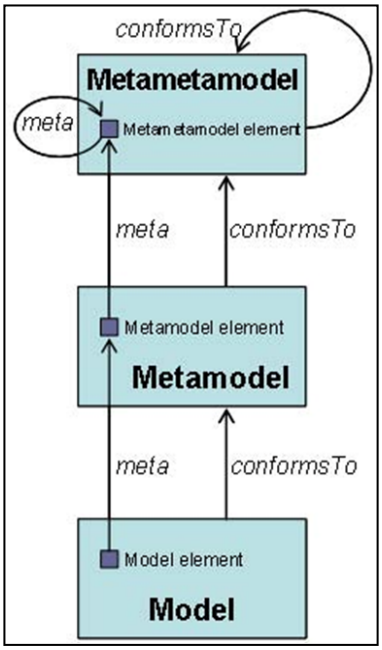

Figura 4: Arquitectura de modelos de tres niveles

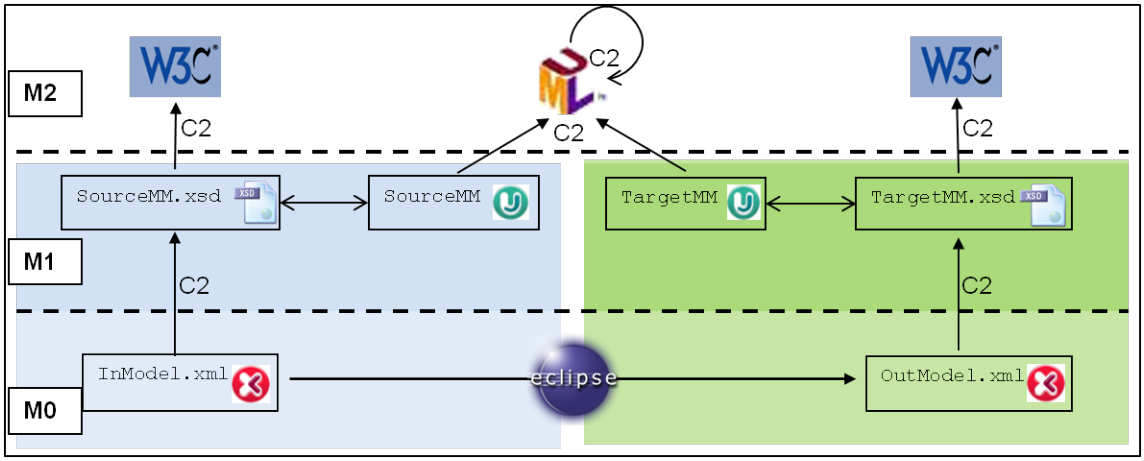

Figura 5: Escenario inicial de transformaciones M2M según la arquitectura de tres niveles

---

Ejemplo. *Se plantea una transformación M2M considerando como meta-modelo origen el meta-modelo MAST2 [7] y como meta-modelo destino el meta-modelo SimMAST, estando ambos descritos empleando UML2 como lenguaje de modelado y también, análogamente, mediante sendas plantillas W3C-Schema que validan respectivamente los modelos de entrada y de salida formulados en XML. Lo siguiente es concebir la manera en que los elementos del meta-modelo origen se transforman en uno o más elementos del meta-modelo destino, lo cual es conveniente plasmar de alguna manera en forma de reglas de transformación, por motivos que posteriormente serán referidos. Una opción podría ser expresar dichas reglas de transformación textualmente y acompañarlo de sendos diagramas de objetos que muestren cómo una instancia de una meta-clase de MAST2 daría lugar a un conjunto de instancias de meta-clases de SimMAST. De aquí en adelante, todo lo relativo a este ejemplo de desarrollo de una transformación M2M se va a mostrar en detalle para el caso de la meta-clase o elemento de modelado Regular_Processor de MAST2, cuya ubicación en dicho meta-modelo se muestra en la figura 6 y su declaración en el schema correspondiente se muestra en File 1.*

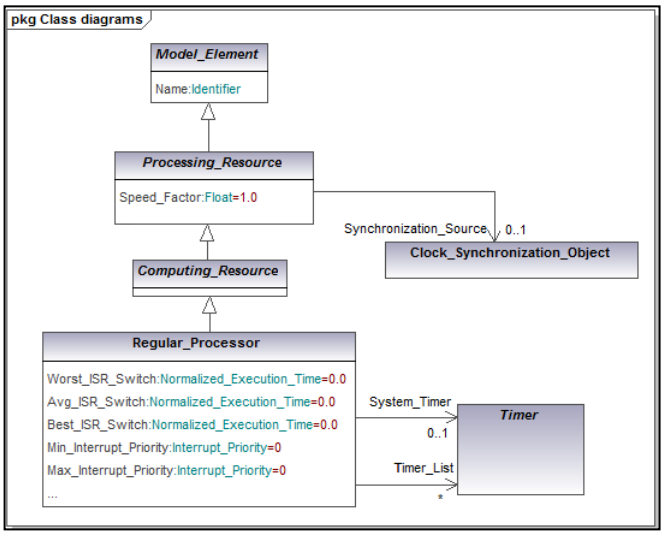

Figura 6: Declaración de Regular_Processor en el meta-modelo MAST2

File: 1: Declaración W3C-Schema de un elemento Regular_Processor de MAST2.

```
<xs:complexType name="Regular_Processor">
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="Timer" type="mast_mdl:Timer_Ref"/>
   </xs:sequence>
   <xs:attribute name="Name" type="mast_mdl:Identifier" use="required"/>
   <xs:attribute name="Speed_Factor" type="mast_mdl:Float" use="optional"/>
   <xs:attribute name="System_Timer" type="mast_mdl:Identifier_Ref" use="optional"/>
   <xs:attribute name="Synchronization_Source" type="mast_mdl:Identifier_Ref" use="optional"/>
   <xs:attribute name="Max_Interrupt_Priority" type="mast_mdl:Interrupt_Priority" use="optional"/>
   <xs:attribute name="Min_Interrupt_Priority" type="mast_mdl:Interrupt_Priority" use="optional"/>
   <xs:attribute name="Worst_ISR_Switch" type="mast_mdl:Normalized_Execution_Time" use="optional"/>
   <xs:attribute name="Avg_ISR_Switch" type="mast_mdl:Normalized_Execution_Time" use="optional"/>
   <xs:attribute name="Best_ISR_Switch" type="mast_mdl:Normalized_Execution_Time" use="optional"/>
</xs:complexType>
```

Para un objeto Regular_Processor de MAST2 se concibe una transformación que lo convierte en un conjunto de objetos que son instancias de meta-clases de SimMAST. La entrada y las salidas de dicha transformación se muestran en las figuras 7 y 8 y su correspondiente formulación XML conforme a los respectivos schemas, para un caso concreto, en los fragmentos de código XML File2 y File 3.

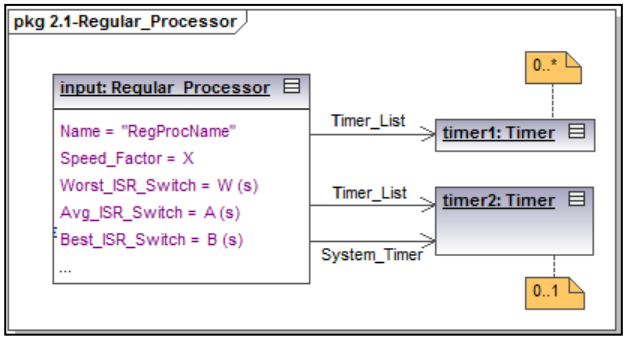

Figura 7: Objeto Regular_Procesor de MAST2

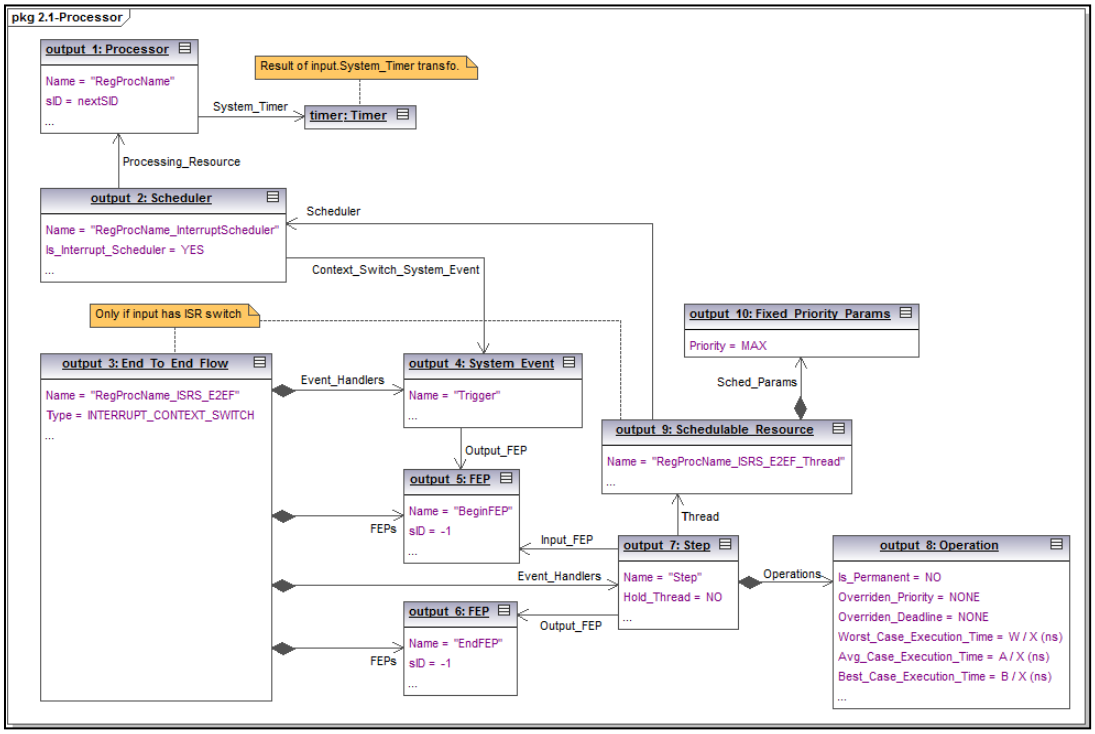

Figura 8: Conjunto de objetos SimMAST en los que se transforma un objeto Regular_Processor de MAST2

File 2: Declaración XML de un elemento Regular_Processor concreto.

```
<mast_mdl:Regular_Processor Name="Proc1"  Speed_Factor="2.0"  System_Timer="Proc1_Ticker"
   Min_Interrupt_Priority="31"  Max_Interrupt_Priority="32"
   Worst_ISR_Switch="15.0E-6" Avg_ISR_Switch="11.0E-6" Best_ISR_Switch="10.0E-6">
   <mast_mdl:Timer Name="Proc1_TimerA"/>
   <mast_mdl:Timer Name="Proc1_TimerB"/>
   <mast_mdl:Timer Name="Proc1_Ticker"/>
</mast_mdl:Regular_Processor>
```

File 3: Declaración XML de la salida en que se transforma el Regular_Processor de File 2.

```
<sim_mm:Processor Name='Proc1' System_Timer='Proc1_Ticker' SID='1'>
   <sim_mm:Percent_On_State/>
</sim_mm:Processor>

<sim_mm:Scheduler Name='Proc1_InterruptScheduler'
   Processing_Resource='Proc1'
   Is_InterruptScheduler='YES'
   ContextSwitch_System_Event='Proc1_ISRS_E2EF.Trigger'/>

<sim_mm:Schedulable_Resource Name='Proc1_ISRS_E2EF_Thread' Scheduler='Proc1_InterruptScheduler'>
   <sim_mm:Fixed_Priority_Params Priority='2147483647'/>
</sim_mm:Schedulable_Resource>

<sim_mm:End_To_End_Flow Name='Proc1_ISRS_E2EF' Type='INTERRUPT_CONTEXT_SWITCH'>
   <sim_mm:System_Event Name='Trigger' Output_FEP='BeginFEP'/>
   <sim_mm:FEP Name='BeginFEP' SID='-1'/>
   <sim_mm:FEP Name='EndFEP' SID='-1'/>
   <sim_mm:Step Name='Step'
      Input_FEP='BeginFEP' Output_FEP='EndFEP' Thread='Proc1_ISRS_E2EF_Thread' Hold_Thread='NO'>
      <sim_mm:Operation
         Worst_Case_Execution_Time='7500' Avg_Case_Execution_Time='5500' Best_Case_Execution_Time='5000'
         Overriden_Priority='0' Overriden_Deadline='0'/>
   </sim_mm:Step>
</sim_mm:End_To_End_Flow>
```

El entorno de desarrollo MAST2 está implementado sobre la plataforma Eclipse [8], y por ello se utiliza una tecnología MDE que sea compatible con este entorno. Llevar a cabo transformaciones M2M –y en general cualquier tarea en el ámbito del DSDM– utilizando Eclipse como entorno de trabajo conduce de forma natural al proyecto constituyente de Eclipse dedicado a tal efecto, esto es, *Eclipse Modeling Project* [9] y a utilizar como marco de modelado uno de sus subproyectos, *Eclipse Modeling Framework* (EMF) [10][11], y en particular, aparte de muchos otros recursos que el *framework* ofrece, su núcleo, el meta-meta-modelo Ecore [12], o lo que es lo mismo, el modelo utilizado en EMF para representar modelos. Sin entrar en detalles, podría describirse Ecore como una especie de UML más ligero, cubriendo únicamente su parte dedicada a los diagramas de clases –en la figura 9 se muestra el *kernel* de Ecore y puede observarse que los nombres de sus clases (meta-clases) se corresponden de manera muy próxima con los términos de UML. De hecho, una pregunta obvia podría ser por qué UML no es el meta-meta-modelo de EMF, esto es, por qué EMF necesita su propio meta-meta-modelo. La respuesta es simplemente que UML en su completitud soporta un nivel de modelado mucho más ambicioso del que se pretende soportar en EMF –UML, por ejemplo, permite modelar el comportamiento de una aplicación y no sólo su estructura de clases– con lo que se ha optado por un subconjunto ligero y simplificado de UML.

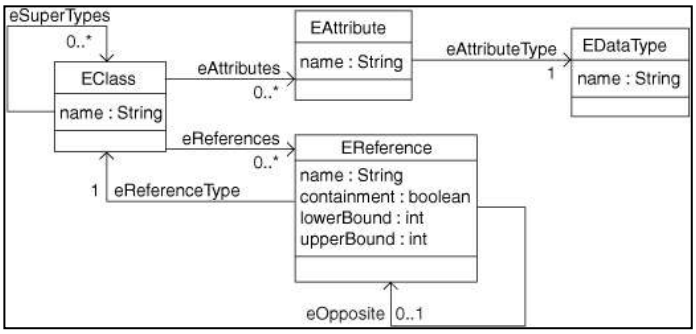

Figura 9: Kernel del meta-meta-modelo ECORE

Elegir como entorno de trabajo para llevar a cabo transformaciones M2M el marco Eclipse/EMF/Ecore conlleva, como primera medida, que sea imprescindible realizar una conversión de los meta-modelos origen y destino formulados en UML a su versión utilizando Ecore como lenguaje de modelado, así como poseer el modelo de entrada también en formato Ecore, y por otro lado, comporta que escoger ATL [13][14][15] como lenguaje de transformación de modelos sea una elección casi obligada. El hecho de que ATL sea un lenguaje de transformación de modelos principalmente declarativo y basado en especificar reglas de transformación aplicadas sobre los objetos del modelo de entrada cuyo tipo sea el declarado en cada una de las reglas explica la recomendación anterior de concebir la transformación global como un conjunto de reglas de transformación.

VI. TRANSFORMACIÓN M2M BAJO EMF/ECORE

Las consecuencias de requerir una formulación XML sobre los modelos de entrada y salida no se limitan a que se duplique el número de artefactos en el nivel M1 del escenario inicial, sino que la transformación M2M considerada desde un punto de vista semántico (del dominio representado por el meta-modelo origen al dominio representado por el meta-modelo destino) se ha de desglosar en una sucesión de tres transformaciones consecutivas, siendo la primera de ellas una transformación de texto a modelo (T2M) y la tercera de ellas una transformación de modelo a texto (M2T), flanqueando ambas transformaciones a la transformación M2M inicialmente considerada.

En la figura 10 se muestra el nuevo escenario en el que desarrollar la transformación M2M central. Como puede observarse, los meta-modelos origen y destino en su formato UML propio del escenario original han sido sustituidos por sus versiones Ecore (*.ecore) y una transformación M2M implementada mediante ATL (*.atl) se aplica sobre el modelo de entrada, conforme al meta-modelo origen, y también formulado según Ecore (*.xmi), para obtener como resultado el modelo de salida (*.xmi), que ha de ser conforme al meta-modelo destino.

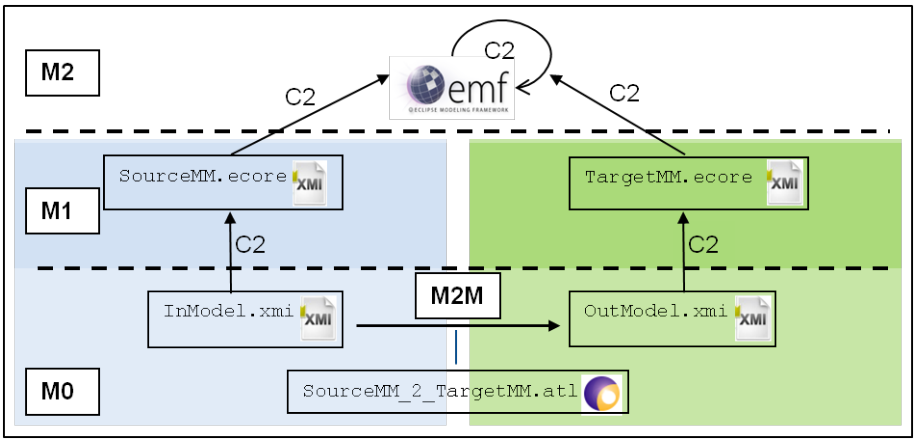

Figura 10: Transformación M2M central

Ejemplo (cont.). *En las figuras 11 y 12 se muestran, respectivamente, los meta-modelos origen y destino, formulados empleando Ecore como lenguaje de modelado. Como se dijo anteriormente, el ejemplo se centra en la clase Regular_Processor de MAST2.*

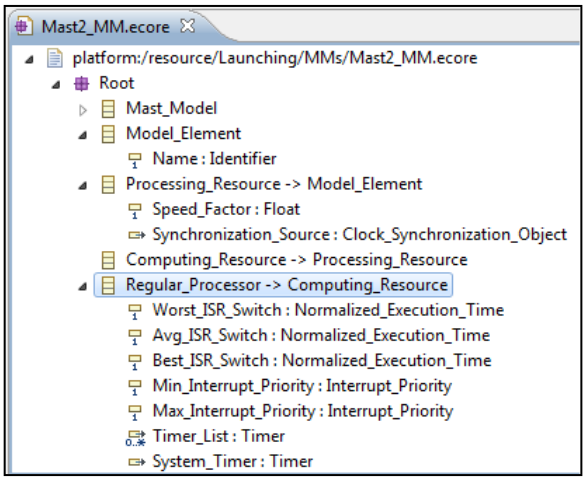

Figura 11: Meta-modelo origen de la transformación M2M

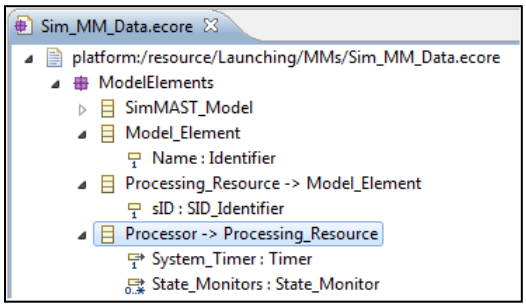

Figura 12: Meta-modelo destino de la transformación. M2M

En File 4 y File 5 se muestran dos fragmentos del fichero ATL que implementa la transformación, en concreto dos de las reglas que actúan sobre objetos del tipo Regular_Processor. Como puede observarse, la transformación se consigue mediante una regla de tipo matched llamada Regular_Processor –File 4– que actúa siempre y reglas lazy –se muestra una de ellas en File 5– que sólo actúan cuando el procesador declara un cambio de contexto por atención a interrupciones no nulo.

File 4: Ejemplo de regla ATL de transformación de Regular_Processor

```
rule Regular_Processor {
    from
        input : Mast2_MM!Regular_Processor
    to
        output_1 : Sim_MM_Data!Processor (Name <- input.Name, sID <- ...
            ...
        ),
        output_2 : Sim_MM_Data!Scheduler (Name <- input.Name + '_InterruptScheduler',
            Is_Interrupt_Scheduler <- true, Processing_Resource <- output_1
            ...
        ),
       ...
}
```

File 5: Ejemplo de regla ATL de transformación cuando hay cambios de contexto.

```
lazy rule Regular_Processor_IRSS_E2EF {
    from
        input : Mast2_MM!Regular_Processor
    to
        output_3 : Sim_MM_Data!End_To_End_Flow (Name <- input.Name + '_ISRS_E2EF',
            Type <- #INTERRUPT_CONTEXT_SWITCH,
            ...
        ),
       ...
}
```

Para que el proceso de transformación planteado inicialmente sea resuelto en toda su magnitud, queda pendiente cómo obtener el modelo de entrada en formato Ecore a partir de su formulación XML y cómo serializar también a XML el modelo de salida producto de la transformación M2M anterior, esto es, establecer el escenario y llevar a cabo las dos transformaciones "periféricas" T2M (de XML a Ecore) y M2T (de Ecore a XML) anteriormente citadas.

VII. TRANSFORMACIÓN DE XML A ECORE

En la figura 13 puede contemplarse el escenario en el que se pone en juego el proceso necesario para transformar el modelo de entrada original formulado en XML a su versión empleando Ecore.

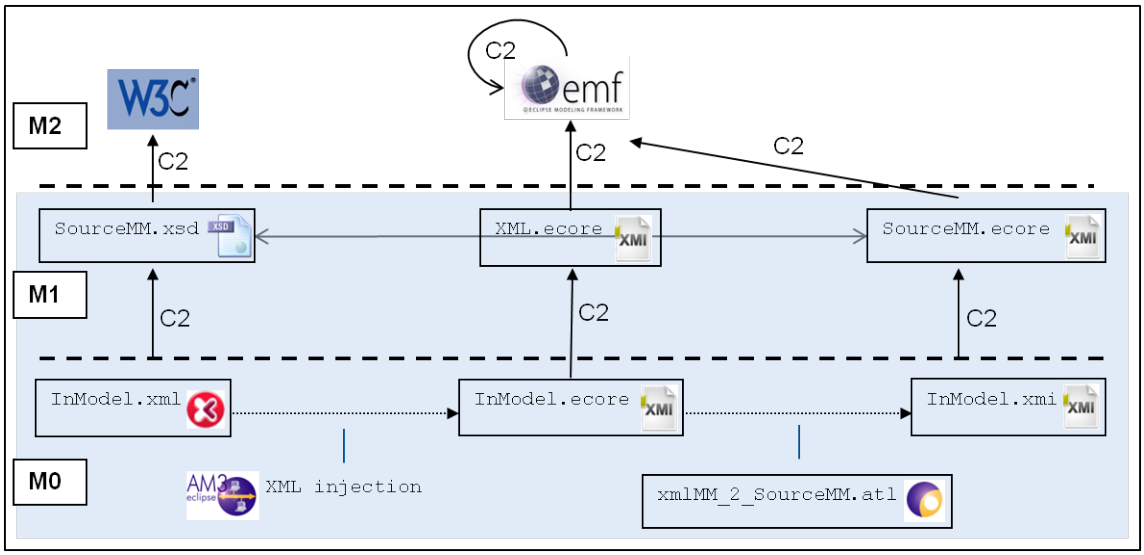

Figura 13: Escenario de transformaciones T2M.

Lo que inicialmente se consideraba como, en teoría, una transformación T2M fruto de desglosar la transformación primigenia M2M en una concatenación de tres transformaciones, se desglosa a su vez en una nueva cadena de dos transformaciones, T2M y M2M. En la primera de ellas, el documento XML que representa al modelo de entrada es transformado en un modelo intermedio –ya formulado mediante Ecore– llamado "modelo XML" y que es conforme al meta-modelo XML mostrado en la Figura 14. Como puede verse, se trata de un meta-modelo muy sencillo pero que comprende todo lo necesario para expresar la semántica contenida en cualquier documento XML, y se proporciona descrito en Ecore como parte de EMF. Esta transformación de XML textual a modelo se denomina *XML injection* y también está implementada y disponible gracias al proyecto AM3 [16].

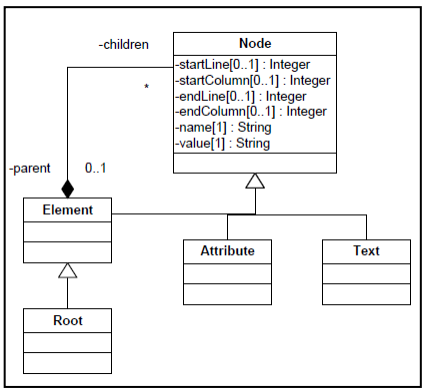

Figura 14: Metamodelo XML

Con el documento XML "inyectado" a Ecore es responsabilidad del desarrollador implementar la transformación M2M que transformará el modelo de entrada conforme al meta-modelo XML –es decir, el modelo intermedio anteriormente citado– en otro con idéntica semántica pero conforme al meta-modelo origen. Esta transformación, aunque de un grado de laboriosidad variable según el caso, es conceptualmente inmediata con un buen conocimiento del W3C-Schema subyacente, frente al cual ha de validar el modelo de entrada en su forma inicial de documento XML, y una vez implementada supone un importante activo en cuanto a futura reutilización.

VIII. SERIALIZACIÓN DE ECORE A XML

En la Figura 15 aparece la transformación periférica M2T que representa el tercer y último paso de la sucesión de transformaciones resultado del desglose inicial de la transformación M2M original. Como puede verse, es simétrica respecto a la transformación T2M estudiada en el epígrafe anterior, es decir, se va a desglosar a su vez en una serie de dos transformaciones, M2M y M2T.

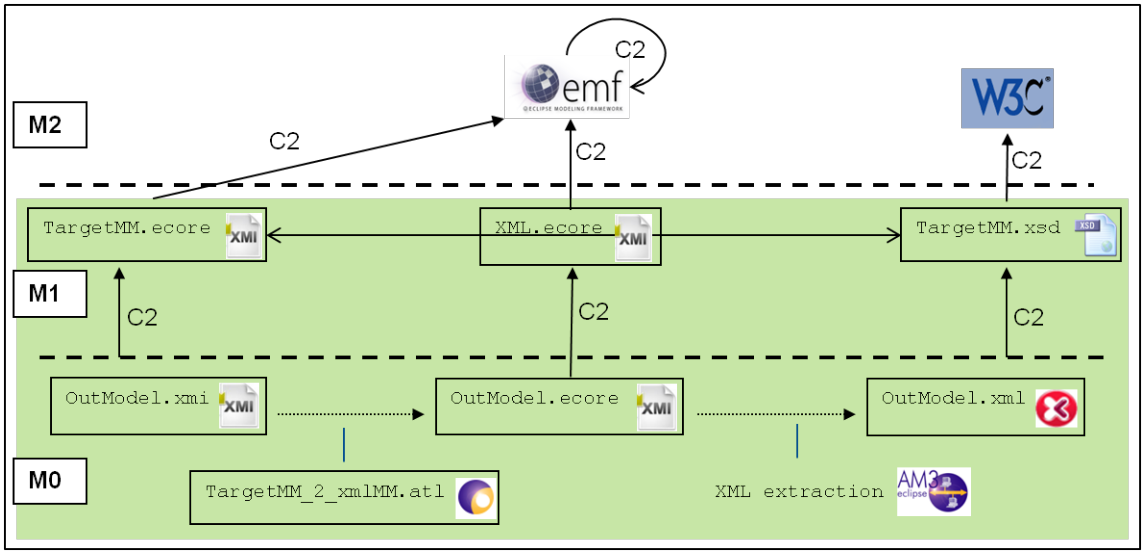

Figura 15: Escenario de transformaciones M2T.

De nuevo, con un buen conocimiento del W3C-Schema frente al que ha de validar el documento XML de salida, es posible implementar una transformación M2M que transforme el modelo de salida en formato Ecore y conforme al meta-modelo destino a otro modelo intermedio también en formato Ecore y conforme al meta-modelo XML de la Figura 14. Las mismas consideraciones anteriores sobre grado de laboriosidad y reutilización se aplican también a esta transformación M2M. Por último, el modelo XML de salida en formato Ecore se puede serializar –proceso conocido como *XML extraction*– a documento XML de forma automática y con ello concluir todo el proceso de transformación originalmente requerido. En la Figura 16 se muestra una representación global de todo el proceso.

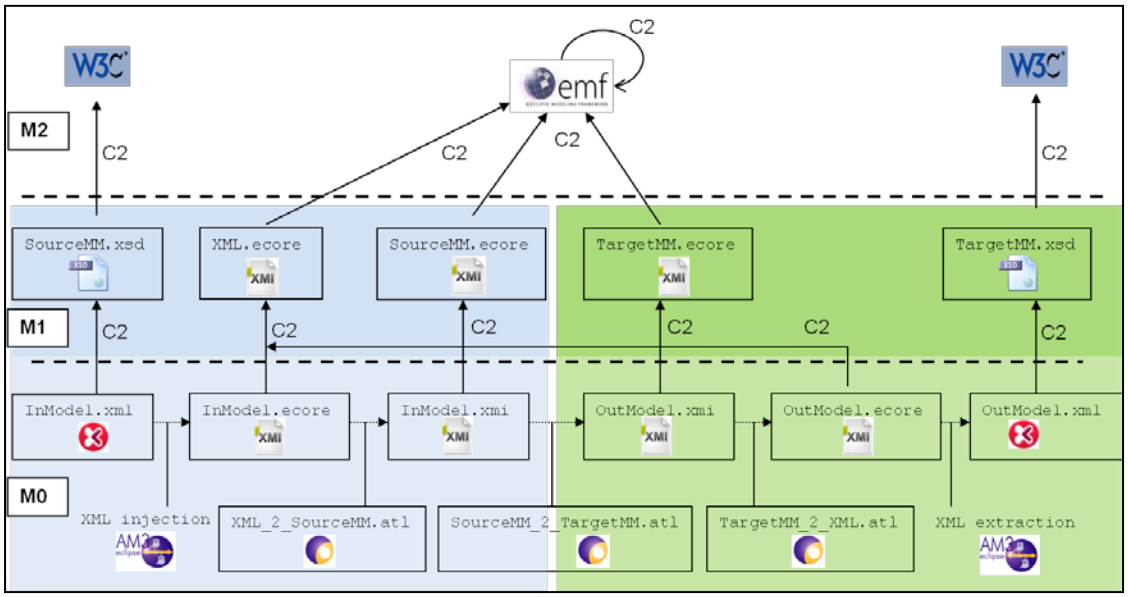

Figura 16: Visión global del proceso

IX. SIGLAS Y ACRÓNIMOS

AM3 …… AtlanMod MegaModel Management
ATL …… ATLAS Transformation Language
DSDM …… Desarrollo de Software Dirigido por Modelos.
EMF …… Eclipse Modeling Framework
MAST ……. Modeling and Analysis Suite
MDA …….. Model-Driven Architecture
MDE …… Model-Driven Engineering
MDD …… Model-Driven Development
M2M …… Model to Model
M2T …… Model to Tex
OMG …… Object Management Group.
T2M …… Text to Model
UML …… Unified Modeling Language
W3C …… World Wide Web Consortium
XML …… Extensible Markup Language

X. REFERENCIAS

[1] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[2] J.Bézívin, S. Gerard, P.A.Muller, L.Rious:"MDA Components:Challenges and Oportunities" In: Metamodelling for MDA.

[3] T. A. Henzinger and J. Sifakis. The Discipline of Embedded Systems Design. *Computer*, 40(10):32–40, 2007.

[4] Gabriel A. Moreno and Paulo Merson:"Model-Driven Performance Analysis"Proc. of the Fourth International Conference on the Quality of Software Architectures (QoSA 2008). Germany, October 14-17, 2008.

[5] http://www.omg.org/spec/UML/2.0/

[6] http://www.w3.org/XML/Schema

[7] C. Cuevas, J.M. Drake et all, “MAST 2 Metamodel” http://mast.unican.es/simmast/MAST_2_0_Metamodel.pdf.

[8] http://www.eclipse.org/

[9] http://www.eclipse.org/modeling/

[10] http://www.eclipse.org/modeling/emf/

[11] EMF: Eclipse Modeling Framework. Dave Steimberg, Frank Budinsky, Marcelo Paternostro y Ed Merks. Dec 16, 2008, Addison-Wesley Professional.

[12] http://www.eclipse.org/modeling/emf/?project=emf

[13] http://www.eclipse.org/atl/

[14] Fréderic Jouault, Freddy Allilaire, Jean Bézívin, Ivan Kurtev and Patrick Valduriez. 2006. ATL: a QVT-like transformation language. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA '06).

[15] Jouault, Frédéric, Kurtev, Ivan. Transforming Models with ATL. Satellite Events at the MoDELS 2005 Conference

[16] http://www.eclipse.org/gmt/am3/