SciVerse ScienceDirect
Disponible en www.sciencedirect.com

Revista Iberoamericana de Automática e Informática industrial 10 (2013) 216–227

www.elsevier.es/RIAI

Beneficios que aporta la metodología MDE a los 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, Avda. Castros s/n, 39005, Cantabria, España.

Resumen

Se analizan los beneficios que aporta el empleo de la metodología *Model-Driven Engineering* (MDE) cuando se utiliza como base y soporte de entornos de diseño de sistemas de tiempo real. Con esta metodología, se incrementa la facilidad de uso del entorno para el diseñador, ya que, en función del paradigma que utilice y de la fase del proceso de diseño que esté llevando a cabo, el entorno le ofrece una vista especializada del sistema, la cual presenta de una forma precisa y coherente la información sobre la que está decidiendo. Por otro lado, una infraestructura MDE facilita el desarrollo de herramientas y su integración en el entorno, ya que, a través de los mecanismos de gestión y transformación de modelos con los que está dotada, cada herramienta recibe únicamente la información que utiliza, estructurada de modo adecuado, y así mismo, puede generar sus resultados de la forma que le sea natural, ya que el entorno los sabe gestionar. Por último, la metodología MDE permite considerar el entorno de diseño de tiempo real como una vista especializada que se integra coherentemente en un entorno más general, el cual soporta las restantes fases del diseño del sistema.

Palabras Clave:
Entorno de diseño, tiempo real, MDE, modelado, herramientas de planificación, reserva de recursos.

1. Introducción

En la última década, la metodología y tecnologías *Model-Driven Engineering* (MDE) (Schmidt, 2006), en particular la visión propuesta por el *Object Management Group* (OMG), conocida como iniciativa *Model-Driven Architecture* (MDA) (Bézivin et al., 2003), se han constituido como el paradigma de referencia para los entornos de diseño y desarrollo utilizados en la ingeniería software. El principio de la metodología MDE consiste en formalizar toda la información que se gestiona en los entornos como modelos, cuya estructura está especificada a través de meta-modelos, y concebir el procesamiento de la información como transformaciones entre modelos, las cuales también están formalmente descritas en forma de modelo. La rápida implantación de la metodología MDE ha sido resultado de los beneficios que se obtienen de ella:

Mayor nivel de abstracción: centra la atención y el razonamiento en los modelos (y no en los datos, código fuente o algoritmos, como ha sido hasta ahora tradicional), lo que representa un nivel de abstracción más alto en la descripción de los sistemas y de los procesos de diseño, posibilitando al diseñador humano una mejor comprensión y gestión de los mismos.

Independencia de la plataforma: hace posible razonar con independencia de la plataforma del entorno de desarrollo que se utilice o de la plataforma en que se vaya a ejecutar la aplicación resultante, lo que facilita la inter-operatividad entre entornos y la reusabilidad de los resultados.

Descripción de múltiples aspectos: permite describir un sistema con tantos modelos como sea necesario, constituyendo cada uno de ellos una vista independiente de alguno de sus aspectos, aunque como todas ellas corresponden a un mismo sistema, el entorno debe mantener la coherencia entre la información que contienen. La multiplicidad de los modelos es clave en los entornos de desarrollo software, ya que cada aspecto de un sistema (especificación funcional, diversas especificaciones no funcionales, despliegue, configuración, empaquetamiento, generación de código, etc.) puede describirse de forma independiente y con la riqueza de detalle que se necesite, pero manteniendo las correspondencias y pasarelas entre ellos.

Lenguajes declarativos para el procesado de la información: ofrece lenguajes específicos para codificar las transformaciones entre modelos (por ejemplo *Query-View-Transformation* (QVT) (OMG, 2008), *ATLAS Transformation Language* (ATL) (Jouault et al., 2006), etc.). Estos lenguajes son de naturaleza declarativa y mediante ellos se implementan las transformaciones en base a los

* Autor en correspondencia
Correos electrónicos: cuevasce@unican.es (César Cuevas), barrosl@unican.es (Laura Barros), lopezpa@unican.es (Patricia López), drakej@unican.es (José M. Drake)

http://dx.doi.org/10.1016/j.riai.2013.03.011

meta-modelos que definen el contenido y la semántica de los modelos que procesan, siendo su formulación muy próxima a cómo el diseñador humano concibe la transformación. La principal ventaja del uso de estos lenguajes es que su formulación es muy estable frente a los cambios de mantenimiento y evolución de los entornos, ya que al plantearse los lenguajes en base a los meta-modelos, los cambios en los códigos de las transformaciones tienen un alcance delimitado e incluso pueden ser asumidos sin necesidad de modificar la propia transformación.

Herramientas reutilizables: las tareas de gestión formal de la información (introducción, visualización, verificación de corrección, análisis de coherencia, integración, partición, almacenamiento y recuperación, etc.) pueden ser formuladas en base a los meta-meta-modelos, lo que hace que las herramientas que las realizan tengan capacidad de operar sobre cualquier modelo con independencia de su naturaleza, y por ello son proporcionadas como parte de la infraestructura. Estas herramientas básicas pueden ser integradas en las herramientas específicas del entorno para llevar a cabo la mayoría de las etapas de gestión de la información, con lo que las herramientas propias del entorno resultan muy simples, al tener que implementar sólo su algoritmo de transformación específico.

Meta-herramientas: las transformaciones de modelos pueden ser ellas mismas especificadas formalmente como un modelo. Esto facilita el desarrollo de meta-herramientas, esto es, herramientas cuya función es generar nuevas herramientas más específicas respecto al sistema que se considera. En la metodología MDE, una meta-herramienta es una transformación más entre el modelo o meta-modelo al que se adapta la herramienta genérica y el modelo que especifica la funcionalidad de la misma.

Por estos beneficios, la metodología MDE se aplica con gran éxito en diferentes tareas de la ingeniería software, como por ejemplo, en la generación de código a partir de modelos, en el desarrollo, ensamblado, configuración y despliegue de tecnologías de componentes, en el desarrollo de pasarelas para la inter-operatividad entre entornos de diferentes fabricantes, etc. También se ha aplicado la metodología MDE al diseño de sistemas embebidos y de tiempo real (Balasubramanian et al., 2006; Gokhale et al., 2007; Henzinger & Sifakis, 2007; Moreno & Merson, 2008), si bien de forma más limitada y en muchos casos como pruebas de concepto.

Los entornos de diseño de sistemas de tiempo real han estado siempre basados en modelos, pero hasta muy recientemente ha sido un mundo cerrado sobre sí mismo. El modelo de tiempo real ha sido considerado la base de los procesos de especificación, análisis, diseño y generación del código de los sistemas de tiempo real. Ejemplo de esta visión cerrada es la especificación *Schedulability, Performance and Time* (SPT) (OMG, 2005) de OMG, cuyo objetivo era la normalización de los entornos de diseño de tiempo real. Fue formulada como una especificación aislada de cualquier otro proceso de diseño de software, por lo que quedó en poco tiempo superada ante el incremento de la complejidad de los sistemas de tiempo real que actualmente se desarrollan, en los que el análisis y diseño de planificabilidad es sólo una parte que concierne a la configuración de sistemas, y cuya arquitectura software ha sido condicionada por su funcionalidad y por el paradigma de diseño requerido para abordar su complejidad (orientación a objetos, ensamblado de componentes, reserva de recursos, particionado temporal, etc.). Este problema ha sido superado con la especificación *Modeling and Analysis of Real-Time and Embedded systems* (MARTE) (OMG, 2011a) realizada posteriormente por OMG, en la que el tiempo real aparece como una vista más del desarrollo de un sistema, integrada con otras muchas vistas (funcionalidad, gestión de recursos, consumo de energía, comportamiento temporal, etc.). Durante los últimos tres años, nuestro grupo de investigación ha participado en el proyecto RT-Model (RT-Model, 2009), cuyo objetivo ha sido probar y evaluar la metodología MDE como base de los futuros entornos de análisis, diseño y desarrollo de sistemas de tiempo real. Así mismo, se ha propuesto la nueva versión del entorno *Modeling and Analysis Suite for Real-Time Applications* (MAST 2) (Cuevas et al., 2012) que, a diferencia de su versión anterior (González Harbour et al., 2001), ha sido concebida para servir de base a la utilización de la metodología MDE. En este artículo se describen los principales resultados y conclusiones obtenidos en estos proyectos.

El artículo se organiza en cinco secciones que se desarrollan tras esta introducción. En la sección 2, se revisan los aspectos de los entornos de desarrollo de tiempo real que pueden ser mejorados aplicando la metodología y las tecnologías MDE. Se analizan tanto la forma de dar soporte a través de modelos coherentes a la información que requiere el entorno, como a la implementación a través de procesos de transformación entre modelos de las herramientas requeridas. En la sección 3 se describe un entorno especializado en el diseño de aplicaciones de tiempo real basadas en reserva de recursos, como ejemplo de entorno basado en la metodología MDE. En la sección 4 se describen, también a modo de ejemplo, diferentes recursos y herramientas que son de utilidad en cualquier entorno de diseño de tiempo real basado en MDE. Por último, en la sección 5 se describen las principales conclusiones que se obtienen y el futuro que prevemos para los entornos y tecnología que han sido propuestos.

2. Entornos de desarrollo de sistemas de tiempo real y tecnologías MDE

Tradicionalmente, los entornos de desarrollo de sistemas de tiempo real han estado orientados al análisis y configuración de la planificabilidad del sistema concreto. Para ello, se utiliza una información específica que describe la temporalidad de las respuestas reactivas que constituyen la actividad del sistema en la plataforma en que se ejecuta, y de los recursos protegidos, activos y pasivos, requeridos para su ejecución, que al tener que ser utilizados con exclusión mutua en la ejecución de las respuestas, introducen tiempos de bloqueo y suspensión añadidos que retrasan las ejecuciones. En base a esta información, las herramientas del entorno aplican análisis de planificabilidad de peor caso y determinan los parámetros de configuración que deben ser asignados a los recursos planificables para que todas las respuestas se ejecuten antes de que se alcancen los plazos que tengan establecidos. Bajo este escenario no se utiliza ninguna relación formal entre el modelo de tiempo real en el que se basa el diseño y el sistema que se ejecuta. Los resultados del análisis se transfieren directamente al código fuente de la aplicación, y en algunos casos más complejos, se transfieren como parámetros de configuración en el lanzamiento de la aplicación.

Este tipo de entorno no es adecuado para el desarrollo de los sistemas de tiempo real que actualmente se utilizan. Las aplicaciones son cada vez más complejas y se construyen con diferentes estrategias de modularización (objetos, particiones, componentes, etc.). Las plataformas en que se ejecutan pueden ser distribuidas, heterogéneas, basadas en particionado temporal, etc.

Su configuración puede admitir diferentes opciones y estar automatizada, y su despliegue en la plataforma puede estar especificado a través de planes de despliegue. Estas características requieren entornos de desarrollo más complejos, basados en múltiples modelos complementarios que describan todos los aspectos del sistema, y cuyas herramientas se encarguen no sólo de realizar el análisis de planificabilidad sino también de garantizar en todo momento la coherencia de los modelos.

El número y la diversidad de modelos en los entornos de diseño de sistemas de tiempo real es debida a la necesidad de satisfacer de forma independiente dos puntos de vista complementarios:

1. El punto de vista del desarrollador de aplicaciones de tiempo real que hace uso del entorno. Éste requiere modelos en los que se utilicen elementos que correspondan a los entes 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:
   - En el caso del diseñador de un sistema embebido de tiempo real simple, el modelo adecuado sería el que se basa en entes de modelado de bajo nivel, tales como *threads* y *end-to-end-flows*, operaciones, mutexes, etc.
   - En el caso de que se utilicen paradigmas de diseño modulares (orientación a objetos, componentes, etc.), los elementos de modelado deben corresponder a conceptos de más alto nivel de abstracción, tales como componentes, objetos (activos, pasivos y protegidos), servicios de middleware, planes de despliegue, etc.
   - 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. El punto de vista del desarrollador de herramientas de análisis y diseño de planificabilidad, las cuales constituyen el núcleo fundamental del entorno de desarrollo. Éste requiere disponer de modelos mucho más simplificados, con sólo los elementos y asociaciones que corresponden a la abstracción que procesan las herramientas. Por ejemplo:
   - En el caso de las herramientas de análisis de planificabilidad, la formulación del modelo de acuerdo a un diseño reactivo que describa las transacciones que se ejecutan, el modelo de carga y los requisitos temporales, así como un modelo de recursos que describa los retrasos por suspensión y bloqueo que ocurren como consecuencia del acceso a los mismos con exclusión mutua.
   - En el caso de las herramientas de análisis de planificabilidad basado en reserva de recursos, los niveles de concurrencia, el modelo de carga, los requisitos temporales, etc.
   - 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 empleo de las tecnologías MDE incrementa 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 dichos modelos y del mantenimiento de la coherencia entre ellos.

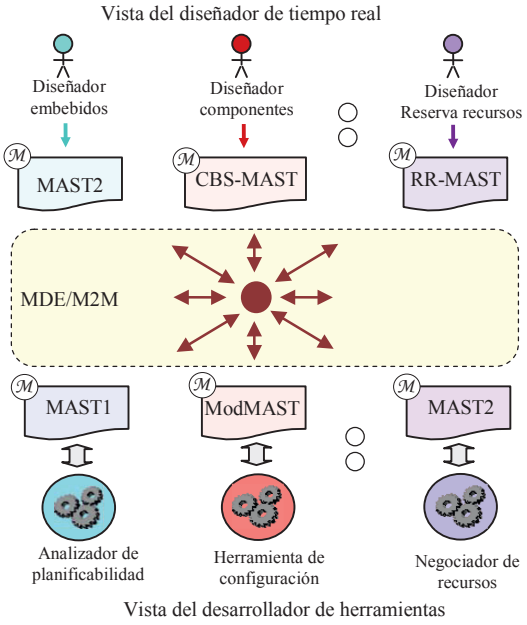

Figura 1. Papel de la metodología MDE en los entornos de tiempo real.

Además de hacer posible que en un entorno de desarrollo de tiempo real se puedan proporcionar modelos adaptados a cada punto de vista o nivel de abstracción, existe un conjunto de operaciones de gestión de la información que son independientes de su naturaleza y a los que la metodología y tecnologías MDE proporcionan soporte a muy bajo costo. Su uso generalizado en cualquier ámbito de interacción con el usuario proporciona una gran facilidad de uso al operador, el cual reconoce en el entorno mecanismos que le son familiares. Así mismo, todas estas tareas de gestión proporcionadas por la plataforma MDE simplifican el desarrollo de las herramientas específicas, ya que simplemente las usan y no deben incluirlas como parte de ellas. Los siguientes son ejemplos de recursos típicamente disponibles en una infraestructura MDE y de tareas que son fácilmente realizables con la metodología MDE, que por tanto simplifican el uso del entorno de desarrollo de tiempo real tanto a desarrolladores de aplicaciones como a desarrolladores de herramientas:

1) *Entorno gráfico amigable.* La interfaz gráfica del entorno, con la que el usuario trabaja, es un aspecto clave del mismo. A través de ella, el operador gestiona los datos y lanza la ejecución de las herramientas. El entorno debe ser ergonómico, capaz y extensible, pero su característica más relevante es que el operador lo encuentre conocido, y sin más instrucciones, sea capaz de manejarlo. Ésta es una de las principales ventajas que tiene el uso de un entorno basado en una infraestructura MDE: su uso generalizado en entornos de ingeniería software lo hace completamente familiar para cualquier usuario. Por ejemplo, si se utiliza como base una plataforma como Eclipse, la interfaz de usuario para el entorno de desarrollo se genera definiendo una perspectiva específica que incorpore los elementos de interacción que se necesiten, y enlazando a ellos las invocaciones de las herramientas genéricas y específicas que se consideran útiles.

Cualquier usuario que lo utilice, encontrará de forma intuitiva las zonas de gestión de ficheros, de edición, de navegación, de lanzamiento de herramientas o de visualización de resultados.

2) *Introducción y modificación de datos*. Son tareas que el operador tiene que realizar muchas veces, y por ello es importante que pueda hacerlo a través de una interfaz gráfica que sea amigable para él. Su trabajo se simplifica si la interfaz proporciona ayuda sobre los datos válidos que se pueden introducir. Esta información está disponible en el entorno, contenida en el correspondiente meta-modelo. Como se muestra en la figura 2, en el caso de que se utilice la infraestructura Eclipse/EMF, se dispone de dos editores (*Sample Reflective Ecore Editor* para modelos y *Sample Ecore Editor* para meta-modelos) que permiten mostrar los elementos del modelo en base a la relación de agregación entre ellos (siempre con estructura de árbol), posibilitan agregar cualquier elemento nuevo que el meta-modelo permita y, cuando un elemento esté seleccionado, a través de la ventana de propiedades (*Properties View*) se pueden modificar los valores de los atributos y de las referencias con las restricciones de tipo, rango de valores, multiplicidad, permisos de modificación, etc. que están especificados en el meta-modelo. Esta herramienta es eficiente, amigable y universal, y está siempre disponible para introducir cualquier modelo que se requiera en el entorno.

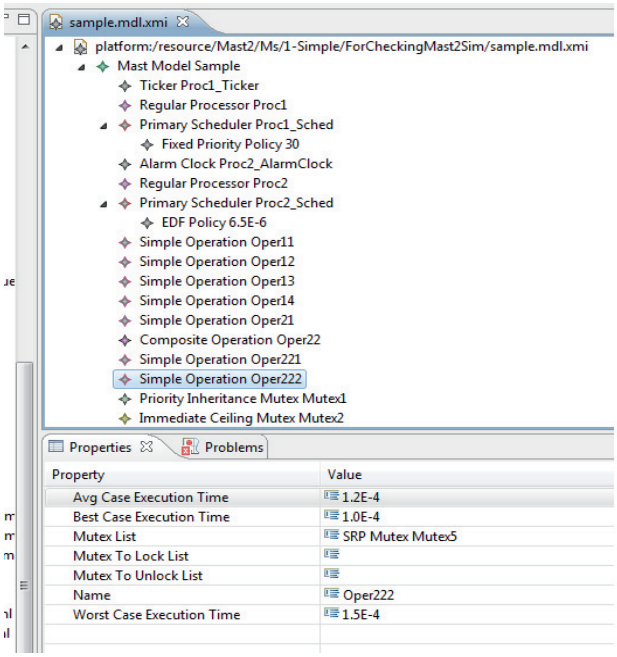

Figura 2. Editor en árbol y ventana de propiedades.

3) *Presentación de resultados*. Cuando el entorno de desarrollo está basado en la metodología MDE, todos los resultados que genera cualquier herramienta también son formulados como un modelo que es conforme a un meta-modelo definido en el entorno. Por tanto, los resultados pueden visualizarse como cualquier otro modelo. Además, el entorno proporciona recursos de visualización tabular que pueden utilizarse de forma sencilla si el modelo de resultados se convierte previamente a través de una transformación M2M al modelo reducido con los campos que se quieren incluir en la visualización.

4) *Almacenamiento y recuperación de la información*. El almacenamiento y recuperación de cualquier modelo se realiza en el formato estandarizado por OMG denominado *XML Metadata Interchange* (XMI) (OMG, 2011b). Por ello, los modelos no sólo son recuperables en el entorno en que se han salvado, sino que pueden ser intercambiados con cualquier otro entorno.

5) *Validación de la información*. Un meta-modelo puede ser definido de forma que cualquier modelo conforme a él siempre posea datos coherentes. Sin embargo, es frecuente que los meta-modelos estén definidos de forma laxa y que los modelos conformes a ellos necesiten una validación posterior para comprobar si sus datos son coherentes. Hay dos razones por las que se definen los meta-modelos como laxos:

- A veces se necesita que la estructura del meta-modelo sea sencilla para que sea fácil su posterior extensión y mantenimiento. Para que un meta-modelo pueda garantizar una coherencia completa en los modelos que son conformes a él, se puede necesitar una estructura interna compleja dotada de una gran número de tipos primitivos en vez de los usuales (int, real, boolean, char, string, etc.). Así mismo, se tiene que definir una jerarquía de herencia de muchos niveles para conseguir especializar totalmente las posibles referencias y especificar sus multiplicidades. Cuando se utilizan meta-modelos laxos las condiciones que garantizan la coherencia pueden especificarse como un conjunto de restricciones sobre el meta-modelo, formuladas a través del *Object Constraint Language* (OCL) (OMG, 2012), también estándar de la OMG.
- Otras veces la validez del modelo no es relativa a su coherencia intrínseca, sino relativa al uso que se va a hacer de él. Por ejemplo, cuando un modelo va a ser procesado por diferentes herramientas dentro del entorno, puede ocurrir que cada herramienta imponga unas restricciones diferentes. En este caso puede ser preferible utilizar un único meta-modelo y definir un conjunto de restricciones OCL por cada herramienta que las requiera.

Un entorno MDE puede ofrecer herramientas genéricas que permitan validar un modelo respecto de un conjunto de restricciones OCL. En este caso, y como se muestra en la Figura 3 para el caso del entorno MAST 2, las herramientas del entorno de desarrollo pueden invocarlas sin necesidad de implementarlas internamente.

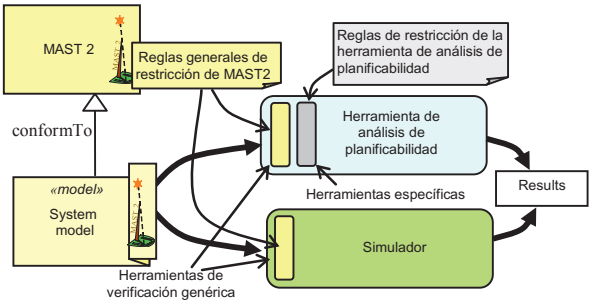

Figura 3. Ejemplo de modelos que requieren validación.

6) *Adecuación de la información y recuperación de los resultados*. Es habitual que las herramientas utilicen conjuntos parciales de los datos de un modelo cuando operan sobre él. Así mismo, los resultados que generan las herramientas pueden ser sólo una parte de los requeridos por el sistema, e incluso estar formulados en base al meta-modelo local a la herramienta, y no respecto del meta-modelo general del sistema con que el

desarrollador los requiere. Como se muestra en la Figura 4 para el caso de MAST 2, el entorno MDE proporciona para estos casos procesos de conversión M2M que permiten adecuar el modelo del sistema (*Scada.xmi*) al modelo requerido por la herramienta (*ScadaSim.xmi*), y de forma coordinada recuperar los datos generados por la herramienta (*ScadaSimRes.xmi*) hacia los datos requeridos por el desarrollador (*ScadaRes.xmi*).

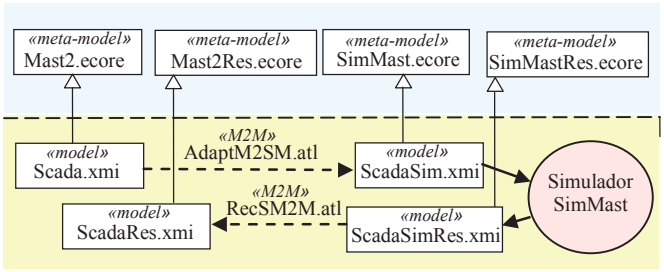

Figura 4. Adecuación a los modelos y resultados de las herramientas.

7) *Importación y exportación de la información:* la plataforma MDE cubre las fases del proceso de desarrollo y en ella toda la información está formulada como modelos respaldados por meta-modelos y codificados de acuerdo con la tecnología que utilice y proporcione su infraestructura de modelado. Pero habitualmente el entorno debe comunicarse con otros ámbitos exteriores a él que no están basados en MDE, como bases de datos, otros entornos de desarrollo, o la propia plataforma de ejecución para la que se desarrolla la aplicación, plataforma en la que han de desplegarse los códigos ejecutables y los ficheros de configuración. En estos casos, hay que inyectar la información a forma de modelo o extraerla a otros formatos no estándares. Actualmente, casi toda la información que se intercambia utiliza formato XML, para el cual la metodología MDE ofrece herramientas genéricas de conversión de texto a modelo (T2M) o de modelo a texto (M2T) que permiten la importación o exportación de datos sin necesidad de implementar generadores específicos.

8) *Integración de herramientas externas:* el entorno Eclipse está desarrollado sobre una plataforma Java y por ello, para el desarrollo de herramientas Java, el entorno da un soporte completo. Sin embargo, cuando las herramientas han sido desarrolladas en otros lenguajes de programación como C++, Ada, Python, etc. en base a conseguir mejores prestaciones o simplemente porque se trata de herramientas legadas, se necesita disponer de interfaces de aplicación específicas que permitan a las aplicaciones acceder a la información de los modelos o crearlos.

3. Entorno de desarrollo de aplicaciones de tiempo real basadas en reserva de recursos

En esta sección se describe un ejemplo representativo de entorno específico de desarrollo de aplicaciones de tiempo real basado en metodología MDE. Se trata del entorno RR-MAST para la configuración y despliegue de aplicaciones de tiempo real estricto diseñadas en base al paradigma de reserva de recursos (Barros, 2012).

Su objetivo es la generación de la información necesaria para ejecutar sobre una plataforma abierta y dotada de un servicio de reserva de recursos una aplicación de tiempo real estricto desarrollada previamente. La estrategia de reserva de recursos se utiliza para hacer posible que una aplicación de tiempo real estricto se pueda ejecutar en una plataforma abierta, es decir, sin que la carga de trabajo que se está ejecutando en el momento en que se despliega la nueva aplicación tenga que ser conocida por el agente que la configura, despliega y ejecuta. Con esta estrategia, la aplicación, desarrollada en una etapa previa, se ejecuta en tres pasos: 1) configuración de la aplicación en el entorno RR-MAST, para que sea planificable en una plataforma virtual que es definida en base a la arquitectura modular de la aplicación, a la fracción de uso de los recursos requerida para planificarla y al despliegue que se va a realizar sobre la plataforma de ejecución; 2) negociación de la implementación de la plataforma virtual resultante con el servicio de reserva de recursos de la plataforma de ejecución, y, en el caso de que la negociación tenga éxito, 3) ejecución propiamente dicha, haciendo uso de los recursos reservados en la negociación.

Los detalles del proceso de planificación de configuración y despliegue (paso 1) se describen en (Barros, 2012) y (Barros et al., 2013), mientras que este artículo se centra en el entorno MDE que lo da soporte, reseñando los principales modelos y herramientas de transformación de modelos involucrados, los cuales se muestran en la Figura 5.

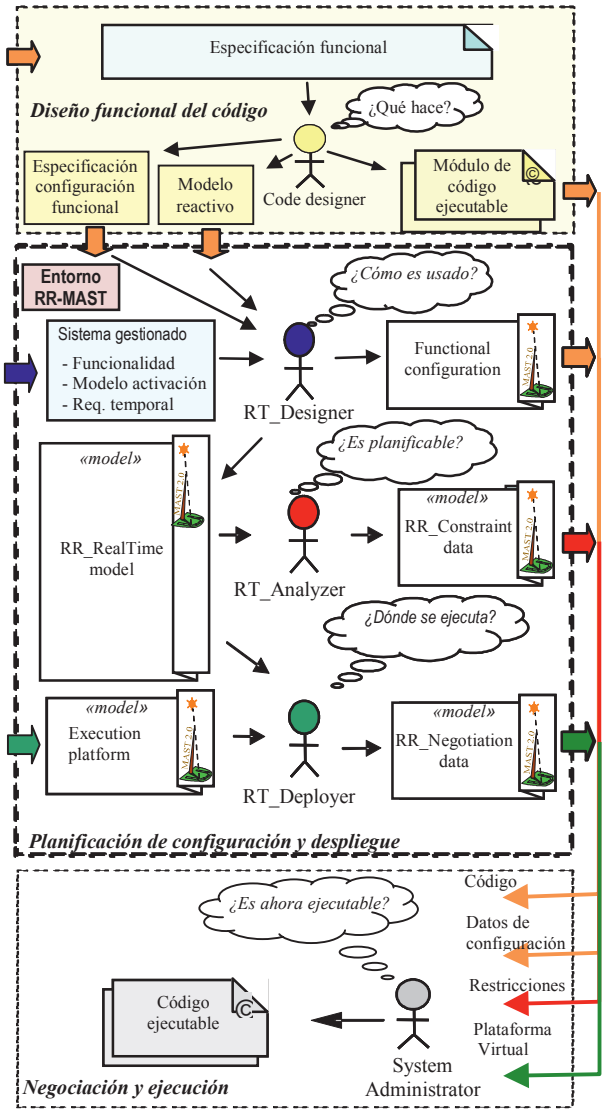

Figura 5. Etapas para el desarrollo y ejecución de una aplicación basada en una estrategia de reserva de recursos.

Como muestra la figura, se considera que la aplicación es desarrollada en una etapa preliminar (*Diseño funcional del código*). En consecuencia, la aplicación se suministra como un conjunto de tres artefactos: 1) El código ejecutable, que no puede ser modificado en el proceso de configuración, despliegue y ejecución, 2) la especificación de los datos de configuración que requerirá el código cuando se ejecute, que lo adaptan a la funcionalidad y despliegue que se le establezca en cada caso concreto, y 3) el modelo reactivo, que proporciona la información sobre su comportamiento temporal necesaria para analizar y configurar su planificabilidad. El objetivo del entorno RR-MAST es tanto presentar al agente encargado de configurar y desplegar la aplicación las opciones de que dispone como recoger sus decisiones, con el propósito de generar tanto los datos de configuración que utilizará el código de la aplicación en su ejecución, como la información que se utiliza en la negociación para la reserva de recursos en la plataforma de ejecución. Cada una de las tres etapas del desarrollo de una aplicación de este tipo requiere un entorno de operación diferente: La etapa de *diseño funcional del código* requiere un entorno de diseño convencional, con herramientas de modelado funcional y arquitectural, generadores de código, etc.; la etapa de *negociación y ejecución* se lleva a cabo sobre la plataforma de ejecución y requiere servicios de reserva de recursos, así como herramientas de despliegue y de ejecución de código, soportadas por el sistema operativo en el caso de plataformas monoprocesadoras o por un middleware en el caso de plataformas distribuidas. En este artículo sólo se describe en detalle la etapa intermedia, relativa a *planificación de configuración y despliegue*, que se realiza en un entorno específico RR-MAST, basado en MDE.

En la Figura 6 se muestran los principales modelos de entrada y salida que corresponden a las actividades de los tres agentes que participan en la etapa de configuración y despliegue, así como las herramientas que utilizan. Además de estos modelos principales, durante esta etapa se gestiona un amplio número de informaciones intermedias, todas ellas formuladas como modelos internos al entorno. En este caso, se ha seguido la estrategia de utilizar únicamente tres meta-modelos:

- *Mast2.ecore*, respecto al cual son conformes los modelos MAST 2 que describen la temporización y la planificación de la aplicación. Este meta-modelo es muy general y permite describir la aplicación en diferentes estados de diseño (modelo reactivo, modelo de la plataforma, modelo basado en plataforma virtual, etc.) y bajo diferentes paradigmas de diseño. En cada caso sólo se utiliza un subconjunto de los elementos del meta-modelo sobre los que se establecen restricciones específicas. Así, un modelo *RR_Reactive* es un modelo conforme a Mast2.ecore y que satisface las restricciones OCL definidas en *RR_Reactive_Constraints*.
- *Impositions.ecore*, respecto al cual son conformes los modelos que establecen las opciones y restricciones que se imponen sobre cualquier meta-modelo al que haga referencia y se utiliza para dirigir la funcionalidad de los asistentes. A veces, el modelo de imposiciones es editado por un agente experto en el sistema, pero en otras ocasiones el modelo es un resultado intermedio generado por alguna herramientas a partir de otros modelos. Por ejemplo, el modelo *Configuration* de la Figura 6 ha sido generado por el diseñador del código para describir las opciones de configuración de la aplicación. En la misma figura, el modelo *Functional configuration* ha sido generado por el asistente *Configuration wizard* en base a las acciones elegidas por el operador.
- *ConstraintViolationDescription.ecore*, respecto al cual son conformes los modelos que describen las violaciones de las restricciones OCL asociadas a cualquier meta-modelo. Como se describe en la sección 4, los modelos que se generan mediante herramientas automáticas (después de que las herramientas hayan sido verificadas) están libres de errores de coherencia "por construcción" y no requieren validación. Sólo los modelos que son inyectados desde fuera del entorno, o aquellos que se construyen con la colaboración del operador requieren validación.

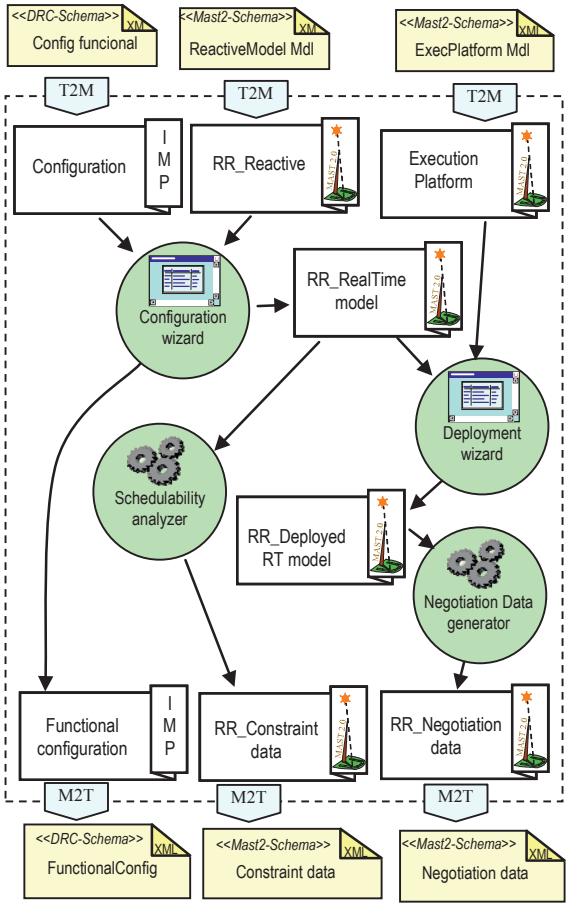

Figura 6. Modelos y herramientas del entorno RR-MAST

La primera etapa de la fase de configuración y despliegue la realiza el diseñador de tiempo real (*RT_Designer*) a fin de adaptar los aspectos configurables de la aplicación a las características del sistema sobre el que va a operar. El entorno proporciona el asistente *Configuration wizard* para ayudar al agente a definir la configuración. Recibe como entrada información que se ha asociado a la aplicación, como meta-datos en la etapa de diseño funcional del código:

1) Modelo *Configuration*: Describe las opciones de configuración que admite la aplicación. En las figuras 7 y 8 se hace referencia al modelo de configuración correspondiente a la aplicación que hemos usado como ejemplo, denominada DRC (*Distributed Robot Control*), y cuya función es implementar un controlador PID de un robot con n ejes. La aplicación permite configurar los ejes del robot que en cada sistema concreto se controlan, así como los parámetros asociados a su control.

2) Modelo *RR_Reactive*: Es un modelo MAST 2 que describe la aplicación desde el punto de vista de su funcionalidad reactiva, esto es, define los tipos de respuestas a eventos que tiene definidos la aplicación, y para cada uno de ellos, describe las actividades que constituyen la respuesta, el patrón de activación y los requisitos temporales que han de ser satisfechos. Es un modelo de tiempo real incompleto (no tiene especificados los recursos de procesamiento sobre los que se ejecutan las actividades de la aplicación, que serán conocidos más tarde cuando se defina su despliegue) y parametrizado, ya que en ciertos casos sólo se definen tipos de transacciones, y no el número y las características concretas de cada una.

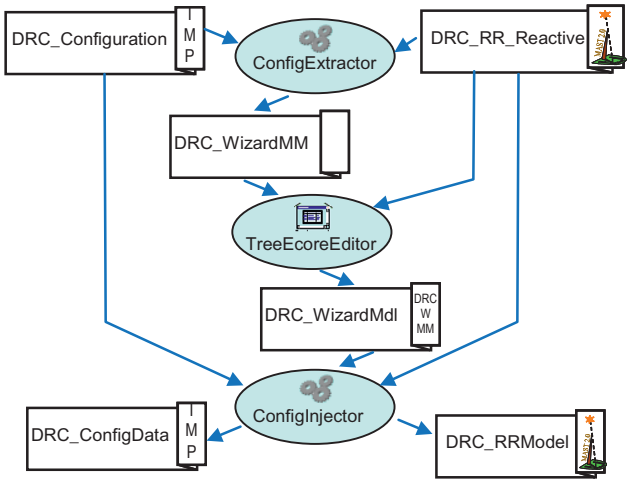

Figura 7. Asistente Configuration wizard.

Como se muestra en la Figura 7, el asistente *Configuration wizard* opera en tres fases: 1) Una herramienta automática (*ConfigExtractor*) interpreta el fichero de configuración (*XXX_Configuration*) que define qué elementos del modelo reactivo son configurables, y crea un meta-modelo temporal (*XXX_WizardMM*) que guía al operador para que introduzca los elementos, atributos y referencias definidos como configurables. 2) El operador hace uso del editor estándar de Eclipse e introduce todos los datos de configuración como un modelo (*XXX_WizardMdl*) que es conforme al meta-modelo previamente generado. Por último, en 3) una herramienta automática (*ConfigInjector*) incorpora los datos de configuración introducidos por el operador al modelo reactivo, generando el modelo *XXX_RRModel*. Además, genera los datos funcionales de configuración (*XXX_ConfigData*) que serán pasados al código de la aplicación en el lanzamiento de su ejecución.

En la Figura 8, se muestra el caso de la aplicación DRC. En la Figura 8.a) se muestra el meta-modelo *DRC_WizardMM*. En él se especifica que la aplicación puede controlar entre 1 y 5 ejes, y por cada eje define las propiedades a las que debe asignarse valor (atributos como *TriggerPeriod* o referencias como *setServoInput*). Así mismo, en la Figura 8.b) se muestra una imagen de los elementos que ofrece el editor estándar de Eclipse al operador. La sección superior permite introducir nuevos elementos (por ejemplo, Axis2) y, una vez seleccionado, la tabla inferior de propiedades permite introducir los valores de los atributos (por ejemplo 0.001 a *TriggerPeriod*) o seleccionar los elementos referenciados (por ejemplo a la referencia *setServoInput* se le permite seleccionar de entre los elementos de tipo *Operation* del modelo *DRC_Reactive*, aquella *setServoInputAxis2* que corresponde al establecimiento de la entrada del servo de ese eje).

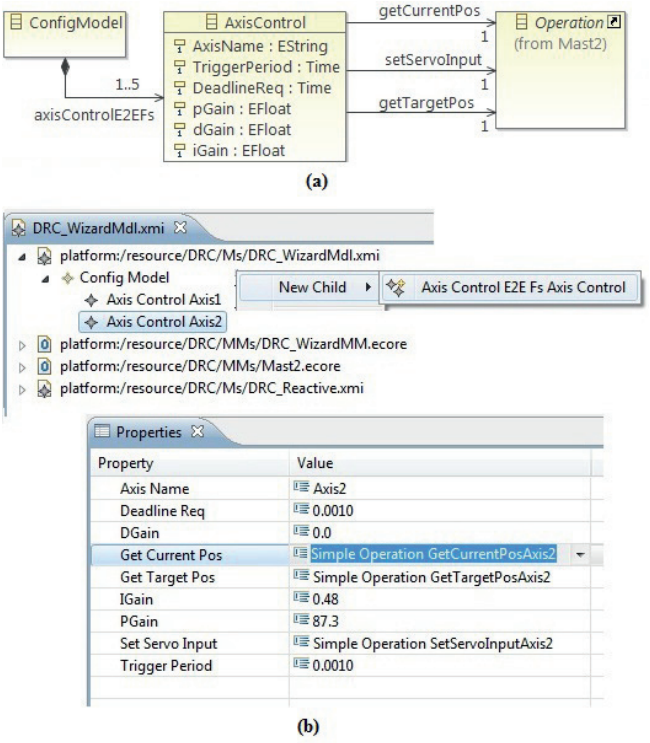

Figura 8. Elementos gráficos del asistente de configuración

MAST 2 permite definir recursos virtuales de planificación, esto es, recursos de planificación que ofrecen dos vistas: la primera vista es convencional y describe los recursos como *threads* o canales de comunicación con parámetros de planificación (prioridad, *deadline*, etc.) determinados, mientras que la segunda vista los describe como recursos virtuales definidos en base a un contrato de uso de recursos basado a su vez en parámetros tales como presupuesto (*budget*), periodo de reposición (*replenishment period*) y plazo (*deadline*). En el modelo *RR_RealTimeModel* se define el modelo temporal de la aplicación en base a una plataforma virtual constituida exclusivamente por recursos virtuales, que especifican las capacidades de procesamiento o anchura de banda de comunicación que requiere la aplicación para ser ejecutada, con independencia de la plataforma de ejecución física que se utilice para su despliegue.

La segunda fase de configuración la realiza el agente *RT_Analyzer*, el cual, sobre el modelo *RR_RealTimeModel*, analiza la planificabilidad de la aplicación en base a la plataforma virtual. Esto lo realiza mediante una herramienta que analiza todas las transacciones del modelo y establece las restricciones que deben cumplirse entre los contratos de uso de recursos de todos los recursos virtuales para que se satisfagan los requisitos temporales establecidos en ellas. El modelo *RR_Constraint_Data* se formula también como un modelo MAST 2 y está compuesto únicamente por transacciones lineales con requisito temporal en su evento de finalización. Esta información es utilizada por el servicio de reserva de recursos en la fase de negociación. El detalle acerca de este proceso de análisis de planificabilidad sobre plataforma virtual se puede consultar en (Barros, 2012).

La tercera fase del proceso de configuración y despliegue la ejecuta el agente *RT_Deployer* a través del asistente *Deployment wizard*. Su función es establecer el plan de despliegue de la aplicación sobre la plataforma de ejecución que se haya elegido. Con este objeto, en un paso previo una herramienta procesa el modelo *RR_RealTime_model* que contiene la información sobre la aplicación, y el modelo *Execution platform* que define la plataforma de ejecución. Esta herramienta genera un modelo intermedio conforme al meta-modelo *Impositions* que contiene todos los recursos virtuales de la aplicación y permite establecer, en cada uno de ellos, la referencia a uno de los procesadores o redes presentes en la plataforma de ejecución. Esta fase da lugar a un nuevo modelo *RR_Deployed_RT* similar al de entrada, pero en el que todos los recursos virtuales tienen ya asignado el procesador o la red de comunicaciones sobre el que se despliegan. Por último, la herramienta *Negotiation data generator*, organiza las plataformas virtuales en base al servicio de reserva de recursos con el que su implementación debe ser negociada, y almacena como un modelo MAST 2 la información necesaria para la negociación.

El entorno descrito en esta sección se ha implementado utilizando la infraestructura que proporciona la plataforma abierta y de software libre Eclipse (Eclipse foundation, a), y en particular su tecnología MDE desarrollada en el subproyecto *Eclipse Modeling Framework* (EMF) (Eclipse foundation, b; Steinberg et al., 2009) del proyecto *Eclipse Modeling Project* (EMP) (Eclipse foundation, c). En ella se utiliza el meta-meta-modelo Ecore definido en él, cuya principal ventaja es su simplicidad (mucho más reducido que el MOF definido por OMG) y que cubre básicamente los aspectos estructurales de los meta-modelos, y sólo residualmente algo relativo al modelado del comportamiento.

Elegir como entorno de trabajo para llevar a cabo transformaciones modelo a modelo (M2M) el ámbito Eclipse/EMF/Ecore, conlleva escoger ATL o QVT como lenguajes de transformación de modelos. En nuestro caso se ha elegido ATL, que es un lenguaje de transformación de modelos muy bien consolidado y que frente a la alternativa QVT proporciona ventajas en la gestión de las restricciones formuladas mediante OCL.

4. Implementación de algunos algoritmos básicos de transformación

En este apartado se describen con mayor detalle algunos de los procesos de transformación que se han desarrollado como parte de la infraestructura MDE de los entornos de tiempo real. Su objetivo no es presentar un catálogo de transformaciones, sino mostrar la estrategia básica que se sigue.

4.1. Transformación de XML a Ecore

El entorno EMF/Ecore de Eclipse no proporciona herramientas estandarizadas para implementar los procesos T2M. En la Figura 9 se muestra la estrategia que se utiliza para transformar una información original formulada en XML a un modelo equivalente formulado conforme a un meta-modelo Ecore. La estrategia que se propone descompone este proceso en el encadenamiento secuencial de dos pasos: el primero, T2M, genérico e independiente del contenido del fichero XML (y por tanto utilizable sin cambio en todos los casos), y el segundo una transformación M2M formulada en lenguaje ATL que sí requiere conocer la estructura de la información del fichero XML y el meta-modelo que define el modelo que se obtiene como resultado.

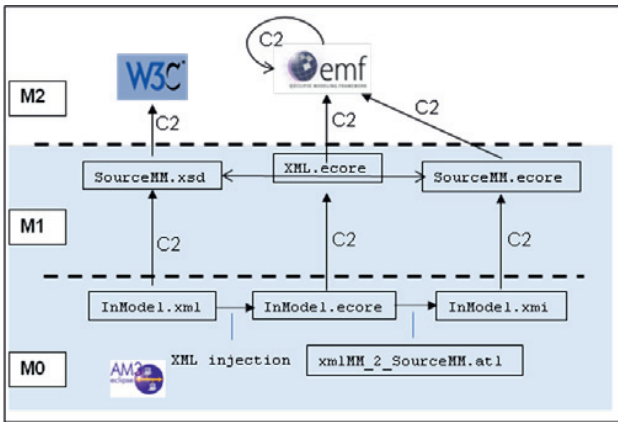

Figura 9. Estrategia de conversión de XML a modelo Ecore

En la transformación T2M, el documento XML que contiene la información de entrada es transformado en un modelo intermedio –ya formulado en base a un meta-modelo Ecore– llamado "modelo XML" y que es conforme al meta-modelo XML mostrado en la Figura 10. 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 inyección XML y también está implementada y disponible gracias al proyecto *AtlanMod MegaModel Management* (AM3) (Eclipse foundation, d).

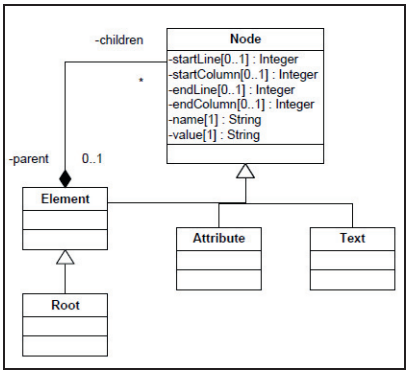

Figura 10. El meta-modelo XML

Con el documento XML "inyectado" a un modelo Ecore, el desarrollador ha de 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 de la estructura del fichero XML que se está transformando.

4.2. Serialización de un modelo Ecore a XML.

En la Figura 11 se muestra la transformación M2T a la que corresponde este caso. Como puede se puede observar, es

simétrica respecto a la transformación T2M estudiada en el epígrafe anterior, es decir, la transformación se descompone en una secuencia de dos transformaciones, M2M y M2T.

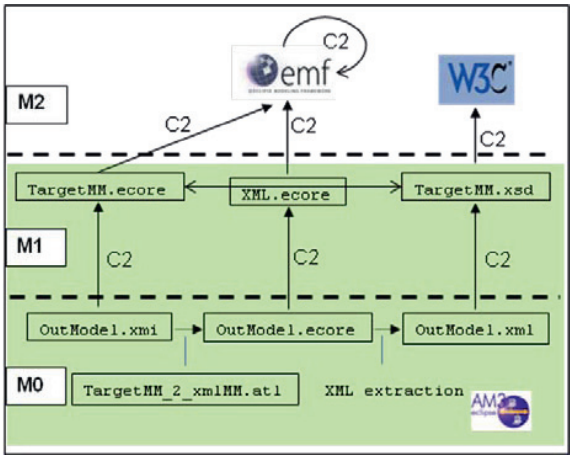

Figura 11. Estrategia de conversión de Ecore a XML

De nuevo, teniendo un buen conocimiento de la estructura de la información en 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 10. 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 extracción XML– a documento XML de forma automática y con ello concluir todo el proceso de transformación originalmente requerido.

4.3. Validación de la coherencia de un modelo especificada a través de un conjunto de restricciones OCL.

En el apartado 2 se indicaron dos razones por las que puede convenir definir meta-modelos laxos que no garanticen la completa coherencia de los modelos que son conformes a ellos. En estos caso, la coherencia de los datos de los modelos se establece a través de conjuntos de restricciones OCL que se asocian bien al meta-modelo, o bien a la herramienta que introduce las restricciones frente a las que se valida el modelo. En estos casos se necesita disponer de un método sistemático que permita validar los modelos que se introducen en el entorno antes de ser procesados.

La estrategia que se propone es construir la herramienta de validación como una transformación M2M que tiene como entrada el modelo a ser validado y como salida un modelo que formula los datos necesarios para describir los incumplimientos de restricciones detectados en el modelo validado. Este modelo-resultado es conforme a un meta-modelo que se denomina *ConstraintViolationDescr* y que se expone a continuación. La estrategia de validar un modelo mediante una transformación M2M definida entre su meta-modelo y un meta-modelo de problemas, obteniendo como resultado un modelo de diagnóstico conforme a éste último, ya fue sugerida en (Bézivin & Jouault, 2006). Ahí se propone un meta-modelo *Problems* muy simple con una única clase con tres atributos *severity*, *location* y *description*. En la presente propuesta la idea se extiende proponiendo un meta-modelo mucho más elaborado que define y jerarquiza la información necesaria para una adecuada descripción de los incumplimientos de restricciones detectados y está dotado de una estructura dirigida a la sistematización de la construcción de la herramienta que realiza la validación.

En la Figura 12 se muestra el meta-modelo *ConstraintViolationDescr* que está siendo utilizado.

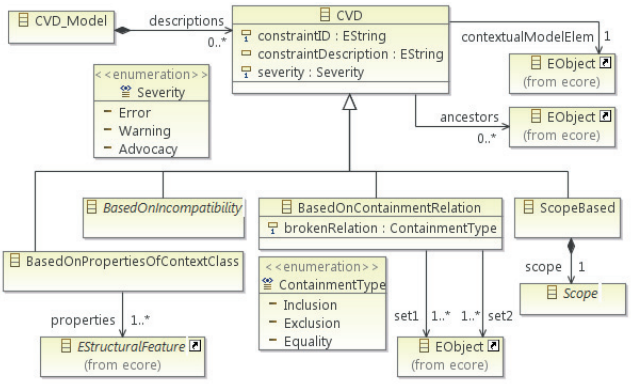

Figura 12. Meta-modelo ConstraintViolationDescr para validar modelos.

Un modelo de descripción de incumplimientos tiene como contenedor principal un elemento del tipo *CVD_Model* que contiene el conjunto de elementos del tipo *ConstraintViolationDescr* que representan cada una de las violaciones detectadas en el modelo validado. Cada objeto de la clase *ConstraintViolationDescr* tiene una referencia *contextualModelElem* que identifica el elemento del modelo de entrada en donde se ha detectado el incumplimiento, un atributo *severity* que cualifica la gravedad de la violación en los niveles *Error*, *Warning* y *Advocacy* y atributos textuales propios de la restricción incumplida, como son su ID y una descripción, que ha de corresponder a un mensaje suficientemente explícito para que un operador humano identifique la semántica de la restricción violada. Además, la referencia *ancestors* representa el conjunto de elementos de modelo que compone la ruta desde el contextual al contenedor principal.

Únicamente con esta clase raíz, representativa de la descripción genérica de incumplimiento, sería suficiente para formular como modelo el conjunto de incumplimientos de restricciones detectados en un modelo, pues la información que describe es apta para cualquier restricción, independientemente de su naturaleza, semántica o formulación OCL. Sin embargo, en el meta-modelo se define toda una jerarquía de subclases de la clase raíz con el objetivo de posibilitar el modelado de descripciones más completas acerca de incumplimientos. Así, en lugar de formular mediante ella las violaciones de cada restricción especificada para un meta-modelo, será posible asociar a cada restricción la subclase que sea más adecuada para describir con mayor nivel de detalle los incumplimientos que de ella se produzcan. Las subclases de primer nivel se muestran en la Figura 12 y se presentan a continuación.

- *BasedOnPropertiesOfContextClass*. Esta clase es adecuada para modelar incumplimientos de restricciones especificadas sobre un conjunto de propiedades (atributos o asociaciones) de la clase contexto de la restricción. Junto a las posibilidades de modelado heredadas de *ConstraintViolation Descr*, la clase *BasedOnPropertiesOfContextClass* define una asociación *properties* que representa punteros a las

propiedades involucradas en la especificación de la restricción.

- *BasedOnIncompatibility*. Esta clase es adecuada para modelar incumplimientos de restricciones consistentes en establecer incompatibilidades entre subclases de dos clases del meta-modelo que se hallan relacionadas con la clase contexto de la restricción a través de sendas cadenas de asociaciones, lo que hace posible considerar que ambas clases están en cierta manera también relacionadas. Se puede dar el caso particular de que una de las clases coincida con la propia clase contexto, en cuyo caso la cadena de asociaciones correspondiente sería nula. *BasedOnIncompatibility* es una clase abstracta que, salvo aportar semántica, no aumenta las posibilidades de modelado heredadas de *ConstraintViolationDescr*. Ello se consigue a través de sus subclases, no detalladas aquí.
- *BasedOnContainmentRelation*. Esta clase es adecuada para modelar incumplimientos de restricciones consistentes en establecer una relación de contención entre dos conjuntos de instancias de una clase que representan cada uno el extremo final de sendas cadenas de asociaciones que partiendo de una misma clase, enlazan ésta con la anterior. Junto a las posibilidades de modelado heredadas de *ConstraintViolation Descr*, la clase *BasedOnContainmentRelation* define dos asociaciones *set1* y *set2* que representan punteros a los elementos de modelo que conforman los conjuntos que incumplen la relación de contención especificada en la restricción.
- *ScopeBased*. Esta clase es adecuada para modelar incumplimientos de restricciones cuya satisfacción depende no sólo del estado de un elemento de modelo, sino también del ámbito o población de elementos de modelo dentro del cual el primero se halla. Junto a las posibilidades de modelado heredadas de *ConstraintViolationDescr*, la clase *ScopeBased* define una asociación *scope*, de tipo composición, que contiene la descripción del ámbito o población de elementos de modelo dentro de la cual la restricción se incumple. En el caso más general, el ámbito puede ser el modelo entero, pero también podría tratarse de ámbitos más específicos, esto es, poblaciones de elementos de modelo que constituyen el extremo de una asociación múltiple.

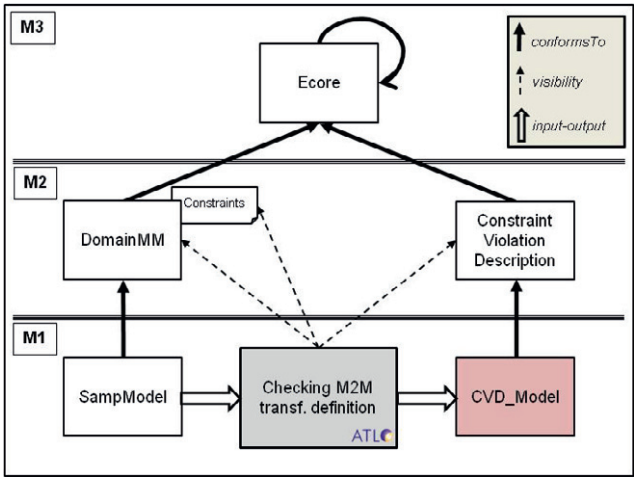

Figura 13. Esquema de una transformación de chequeo

La transformación M2M que convierte el modelo que está siendo validado en un modelo conforme al meta-modelo *ConstraintViolationDescr*, se ha denominado *transformación de chequeo* y constituye la herramienta de validación. En la Figura 13 se muestran los elementos que participan en este proceso.

Para implementar en ATL una transformación de chequeo se ha escogido en este trabajo un estilo basado esencialmente en *helpers* y reglas *called*. Una de las principales ventajas de esta estrategia es que el código ATL resultante tiene una estructura interna muy regular que sigue un patrón uniforme y fácilmente automatizable, lo que puede ser aprovechado favorablemente para desarrollar cada herramienta de validación. Dicha estructura puede comprobarse en el esqueleto de código ATL mostrado en el Listado 1 y se puede descomponer en cuatro bloques, sombreados mediante distintos colores.

Listado 1. Estructura de una transformación de chequeo

```
-- ********************** ATTRIBUTES
-- DomainMM classes (for each Class_J, J = 1, 2, ..., N)
helper def : Class_J : Ecore!EClass = DomainMM!Class_J;
-- DomainMM attributes (for each ClassJ, J = 1, 2, ..., N)
helper def : Class_J_Attr_1 : Ecore!EAttribute = . . .
. . .
helper def : Class_J_Attr_M : Ecore!EAttribute = . . .
-- DomainMM references (for each ClassJ, J = 1, 2, ..., N)
helper def : Class_J_Ref_1 : Ecore!EReference = . . .
. . .
helper def : Class_J_Ref_P : Ecore!EReference = . . .

-- ********************** HELPERS corresponding to the constraints specified for DomainMM
-- For each ClassJ, J = 1, 2, ..., N / ClassJ is context of constraints
-- and for each constraint_JK specified for ClassJ, K = 1, 2, ..., Q
helper context DomainMM!Class_J def : constraint_JK() : Boolean = constraint_JK_OCLexpr;

--- Entrypoint rule of this ATL transfo. It creates a CVD model.
entrypoint rule Create_CVDmodel() {
  using {
    -- For each ClassJ, J = 1, 2, ..., N / ClassJ is context of constraints
    ClassJ_Seq : Sequence(DomainMM!ClassJ) = DomainMM!ClassJ.allInstances();
  }
  to
    output : CVD!CVD_Model (
--      descriptions <- {do section}
    )
  do {
    -- For each ClassJ, J = 1, 2, ..., N / ClassJ is context of constraints
      -- Constraints checking for Class_J
      for (e in Class_J_Seq) {
      -- For each constraint_JK specified for ClassJ, K = 1, 2, ..., Q
      -- Call to the corresponding ckecker helper
        if (not e.constraint_JK())
          thisModule.CalledRuleName(...);
    }
  }
}

-- Rules for creation of CVD instances
rule CVD (...) {...}
rule BasedOnPropertiesOfContextClass (...) {...}
rule BasedOnContainmentRelation (...) {...}
rule ScopeBased (...) {...}
. . .
```

- Bloque I (sección de atributos). Contiene la declaración de atributos (*helper attributes*) correspondientes a las clases y propiedades de clases en el meta-modelo de dominio relacionadas (directa o indirectamente) con la especificación de restricciones sobre tal meta-modelo. Por tanto, el conjunto de restricciones planteadas define completamente este bloque.

▪ Bloque II (sección de *helpers*). Contiene la definición de *helpers* funcionales (uno por cada restricción impuesta sobre el meta-modelo de dominio) que permiten consultar a los elementos del modelo a validar si verifican o no las restricciones especificadas para su tipo (meta-clase). Por tanto, retornan un booleano y cada uno declara como contexto la meta-clase contexto de la restricción correspondiente. Así pues, de nuevo el conjunto de restricciones planteadas define completamente este bloque, siendo su código análogo para diferentes transformaciones de chequeo definidas sobre meta-modelos de dominio diferentes o sobre distintos paquetes de restricciones especificados sobre un mismo meta-modelo.
▪ Bloque III (regla de entrada a la transformación). Es una regla *called* especial, que se invoca en primer lugar al ejecutar la transformación y genera el contenedor principal del modelo de salida, esto es, la instancia *CVD_Model* que ha de albergar las instancias de descripción de los incumplimientos detectados. En la sección *using* de esta regla se declaran variables locales de tipo secuencia para cada una de las meta-clases contexto de restricciones, cada una conteniendo todas las instancias de la correspondiente meta-clase existentes en el modelo de entrada. Estas secuencias se emplearán en la parte imperativa de la regla con el objetivo de ir recorriéndolas para verificar en cada elemento de modelo el cumplimiento de las restricciones pertinentes. El código de la parte imperativa sigue también una estructura muy concreta, utilizándose bucles para recorrer cada secuencia de elementos de modelo y sobre cada uno consultando, gracias al *helper* definido a tal efecto, si NO cumple cada una de las restricciones que le son aplicables, en cuyo caso se genera una instancia de descripción de incumplimiento en el modelo de salida. Para ello se invoca a la regla *called* apropiada de entre las definidas en el bloque siguiente. Por tanto, dada esta estructura tan bien determinada, el código de este bloque también es análogo para toda transformación definida sobre cualquier pareja meta-modelo de dominio + restricciones.
▪ Bloque IV (sección de reglas *called*) Es un bloque fijo, idéntico en cualquier transformación de chequeo (siempre y cuando el meta-modelo destino sea el mismo), y consta de la definición de una regla *called* por cada clase concreta en dicho meta-modelo destino. Cada una de estas reglas está encargada de generar una instancia de tal clase concreta de descripción de incumplimiento e incorporarla al modelo de salida. Aquí se ha optado por asignar a cada regla el nombre de la clase correspondiente.

Por tanto, el código ATL de una transformación de validación tiene una estructura uniforme, con una parte variable (secciones de atributos, *helpers* y regla de entrada) que depende del meta-modelo fuente, pero con un patrón estructural que puede ser fácilmente adaptado y una parte fija (sección de reglas *called*), siempre y cuando se mantenga inalterado el meta-modelo de destino.

5. Conclusiones y trabajos futuros

Para abordar la complejidad que actualmente presenta el desarrollo de las aplicaciones de tiempo real que requiere la industria, se necesitan entornos de desarrollo que den soporte al diseñador y permitan la introducción de todo tipo de herramientas de especificación, modelado, generación de código, despliegue, configuración, análisis de comportamiento temporal, etc. La metodología MDE y las tecnologías que le dan soporte son actualmente la mejor baza de que dispone la ingeniería software para implementar estos entornos. Fundamentalmente por tres ventajas que ofrece:
▪ Eleva el nivel abstracción, centrando la atención y el razonamiento en los modelos, lo que consigue mucha más proximidad a la forma de pensar humana y con ello se facilitan los desarrollos.
▪ Permite manejar en un mismo entorno tantas vista del sistema bajo desarrollo como se necesite. Esto permite integrar en un mismo entorno todos los aspectos que requiere el diseño de software.
▪ Permite la incorporación de herramientas que resuelven de forma genérica y universal la mayoría de las tareas de gestión de la información que requieren los entornos de desarrollo.

Los entornos de desarrollo de sistemas de tiempo real siempre han estado dirigidos y basados en modelos, aunque han permanecido históricamente aislados de otros entornos utilizados por la ingeniería software. La tecnología MDE es una oportunidad de integración y de potenciación de los entornos de desarrollo de tiempo real en otros entornos de desarrollo que tratan otros aspectos que también deben ser gestionados en ellos, a la vista de la complejidad que actualmente alcanzan.

En este trabajo, se han resumido algunas de las experiencias que hemos tenido en la transformación del entorno de diseño de sistemas de tiempo real MAST para que opere en la plataforma genérica MDE proporcionada por *Eclipse Modelling*.

English Summary

MDE technology as support for real-time systems development environments.

Abstract

This paper analyses the benefits provided by the Model-Driven Engineering (MDE) methodology when it is used as the base support for real-time systems design environments. The application of MDE eases the environment usage by the designer, because depending on the chosen paradigm and on the stage of the development process, the environment offers a specialized view of the system, presenting in a precise and coherent fashion the information under decision. On the other hand, an MDE infrastructure facilitates the development of tools and their integration within the environment, as, through the model management and transformation mechanisms this infrastructure is equipped with, every tool receives as input the right information it uses, structured as it needs, and likewise, each tool can generate its results in its natural format, relying in that the environment is able to manage them. Finally, the MDE methodology allows considering the real-time design environment as a specialized view meant to be integrated in a more general framework that encompasses the remaining system design stages.

Keywords:

Design environment, real-time, MDE, modelling and schedulability tools.

Agradecimientos

Este trabajo ha sido financiado por el Gobierno de España bajo las subvenciones TIN2008-06766-C03-03 (RT-MODEL) y TIN2011-28567-C03-02 (HI-PARTES).

Referencias

Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., & Neema, S. (2006). Developing applications using model-driven design environments. *Computer, 39*(2), 33-40.

Barros, L. (2012). Diseño de aplicaciones de tiempo real para plataformas abiertas. Tesis doctoral. Universidad de Cantabria.

Barros, L., Cuevas, C., Martínez, P. L., Drake, J. M., Harbour, M. G. (2013). Modelling real-time applications based on resource reservations. *Article in press, Journal of Systems Architecture, Elsevier.*

Bézivin, J., Gérard, S., Muller, P. A., Rioux, L. (2003). MDA components: Challenges and opportunities. Workshop on Metamodelling for MDA, York, England.

Bézivin, J., Jouault, F. (2006). Using ATL for checking models. *Electronic Notes in Theoretical Computer Science, 152*, 69-81.

Cuevas, C., Drake, J. M., López Martínez, P., Gutiérrez García, J. J., González Harbour, M., Medina, J. L., (2012). *MAST 2 metamodel*

Eclipse foundation. (a). *Eclipse web site.*http://www.eclipse.org

Eclipse foundation. (b). *EMF web site.*http://www.eclipse.org/modeling/emf

Eclipse foundation. (c). *EMP web site.*http://www.eclipse.org/modeling

Eclipse foundation. (d). AM3 *web site* http://www.eclipse.org/gmt/am3

Gokhale, S. S., Vandal, P., Gokhale, A., Kaul, D., Kogekar, A., Gray, J., (2007). Model-driven performance analysis methodology for distributed software systems. *IEEE International Parallel and Distributed Processing Symposium,* 328.

González Harbour, M., Gutiérrez García, J. J., Palencia Gutiérrez, J. C., Drake Moyano, J. M. (2001). MAST: Modelling and analysis suite for real-time applications. *Proceedings of 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands, IEEE Computer Society Press,* 125-134.

Henzinger, T. A., Sifakis, J. (2007). The discipline of embedded systems design. *Computer, 40*(10), 32-40.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P. (2006). ATL: A QVT-like transformation language. *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications,* 719-720.

Moreno, G., Merson, P. (2008). Model-driven performance analysis. *Quality of Software Architectures.Models and Architectures,* 135-151.

OMG. (2005). Formal/05-01-02: UML profile for schedulability, performance, and time, v1.1.

OMG. (2008). *Formal/08-04-03 (meta object facility (MOF) 2.0 query/view/transformation, v1.0)*

OMG. (2011a). *Formal/11-06-03: UML profile for MARTE: Modeling and analysis of real-time embedded systems, v1.1*

OMG. (2011b). *Formal/11-08-10: XML metadata interchange (XMI), v2.4.1*

OMG. (2012). *Formal/12-05-09: Object constraint language (OCL), v2.3.1*

RT-Model. (2009). *Plataformas de tiempo real para diseño de sistemas empotrados basado en modelos. proyecto TIN2008-06766-C03-03*

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer, 39*(2), 25-31.

Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. (2009). In Gamma E., Nackman L. and Wiegand J. (Eds.), *EMF: Eclipse modeling framework* (2nd ed.)Addison-Wesley Longman, Amsterdam, 2nd revised edition (rev). edition.

Apéndice A. Siglas y Acrónimos utilizados en este documento.

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 Text
MARTE….. Modeling and Analysis of Real-Time and Embedded systems
MAST…….. Modeling and Analysis Suite for Real-Time App.
MOF …… Meta-Object Facility
OCL …… Object Constraint Language
OMG …… Object Management Group.
QVT……. Query/View/Transformation
SPT ……. Schedulability, Performance and Time.
T2M …… Text to Model
UML …… Unified Modeling Language
W3C …… World Wide Web Consortium
XMI ……. XML Metadata Interchange
XML …… Extensible Markup Language