KnE Engineering

SIIPRIN Conference Proceedings
Simposio Iberoamericano en Programación Informática
(Ibero-American Symposium on Computer Programming)
SIIPRIN'2017

Knowledge E
Engaging minds

Conference Paper

Software development environments and tools in MDE

Santiago P. Jácome-Guerrero[1], Elizabeth Salazar-Jácome[1], Wilson E. Sánchez-Ocaña[1], Rolando X. Salazar-Paredes[1], and Juan M. Ferreira[2]

[1]Universidad de las Fuerzas Armadas ESPE, Av. General Rumiñahui S/N y Paseo Escénico Santa Clara, 1715231B Sangolquí, Ecuador
[2]Universidad Nacional de Asunción, San Lorenzo, Paraguay

Abstract

Model-Driven Engineering (MDE) is the notion that we can construct a model of a system that we can then transform into the real thing. The development of software in MDE using Domain-Specific Languages (DSLs) has two phases. First, the development of artifacts such as DSLs and transformation mechanisms by the modeling experts. Second, people non-technical experts (domain expert or end user) using the artifacts created develop applications simply because of the high level of abstraction allowed by technology. Several factors are considered to limit the use of MDE. One of them, is the lack of knowledge the tools and the development activities with MDE. To support the MDE initiative, the present work makes a description of the theoretical foundations of MDE, also describes the main activities to build several MDE artifacts with some of the tools most known in this technology.

Keywords: Domain-Specific Language, Model-Driven Engineering, DSL, MDE, activities, tools.

Corresponding Author:
Santiago P. Jácome-Guerrero
psjacome@espe.edu.ec

Received: 28 July 2017
Accepted: 5 September 2017
Published: 30 January 2018

Publishing services provided by Knowledge E

Selection and Peer-review under the responsibility of the SIIPRIN Conference Committee.

1. Introducción

Durante años, el modelado se ha aplicado como una parte importante de desarrollo de software para abordar la complejidad al proporcionar abstracciones de un sistema y ocultar detalles no relevantes [1]. Se desarrollan modelos para representar las varias vistas que puede tener un sistema de software, como sus usuarios y sus interacciones con el sistema, datos, procesos, eventos, interfaces, arquitectura. Los modelos pueden ser construidos de manera manual directamente sobre el papel o mediante sofisticadas herramientas automatizadas, como herramientas CASE (Computer Aided Software Engineering).

Los modelos de la etapa de diseño sirven como entrada en la etapa de implementación para poder codificar la solución en determinado lenguaje de programación de propósito general, como Java, C++. En este punto suele suceder que uno debería modificar algún modelo para hacer viable la implementación o porque se han añadido nuevos requerimientos, sin embargo muchas veces los modelos no son modificados, no reflejando la implementación del código fuente.

La Ingeniería Dirigida por Modelos (MDE) es un enfoque para desarrollar software que vincula directamente el modelo a la aplicación. De esta manera, los modelos no sólo encapsulan el diseño de la aplicación, sino que se utilizan activamente para simular, probar, verificar y generar la implementación del sistema a construir [2]. MDE promete resolver varios problemas a los que se enfrenta al pretender elevar el nivel de abstracción e introducir un mayor grado de automatización al desarrollo de software [3]. El paradigma MDE combina básicamente dos conceptos [4]: a) Lenguajes de Dominio Específico (DSLs), son lenguajes de programación textuales y/o gráficos utilizados para representar aspectos de un determinado dominio, en contraposición a los DSLs, se tiene a los GPLs (Lenguajes de Programación de Propósito General), b) Motores de transformación y generadores, los cuales son empleados para crean varios tipos de artefactos, como otros modelos o como código fuente en un determinado GPL.

La utilización de modelos en el desarrollo de software hace necesario el apoyo de herramientas sofisticadas [5]. Herramientas que deben proporcionar facilidad de uso y la provisión de características de gestión de modelos, elementos considerados como cruciales para una mayor adopción de MDE [1].

MDE puede ser utilizado para desarrollar varios tipos de aplicaciones [2]: a) Ingeniería directa (forward engineering) para crear nuevas aplicaciones, b) Modernización o reingeniería de software (reenginering), c) Models@runtime (sistemas adaptativos).

Actualmente, se considera la existencia de dos tendencias de MDE. La una mediante el empleo de los principios de la Arquitectura Dirigida por Modelos (MDA, Model-Driven Architecture) propuesta por el OMG (http://www.omg.org/) (Object Management Group) y la otra mediante la utilización de DSLs, que se lo referenciará en el presente artículo como enfoque MDE/DSL.

El fundamento principal de MDA es separar la lógica operacional de un sistema de los aspectos de implementación de esta lógica, en una determinada plataforma tecnológica. Esta capacidad de separar los dos aspectos, es posible gracias a las cualidades que tienen los modelos, de poder representar aspectos independientes de un sistema. Por lo que los modelos pueden ser utilizados en todas las etapas del ciclo de vida. MDA

permite especificar tres tipos de modelos [6]. En la fase de análisis se tiene el Modelo Independiente de la Computación (CIM, Computation Independent Model), que se utiliza para describir la funcionalidad de un sistema sin mencionar aspectos de implementación. Posteriormente, se puede iniciar la elaboración del Modelo Independiente de la Plataforma (PIM, Platform Independent Model), el cual permite representar la lógica del sistema, sin entrar en detalles de la tecnología en la cual se la implementará. Por último, se debe elaborar el Modelo Específico de la Plataforma (PSM, Platform Specific Model), el cual hace referencia a consideraciones de la tecnología que se utilizará para la implementación y despliegue de la solución.

Aunque UML es el lenguaje de modelado central de MDA, no todos los modelos tienen por qué estar especificados en dicho lenguaje. La Figura 1 muestra los modelos empleados por MDA.

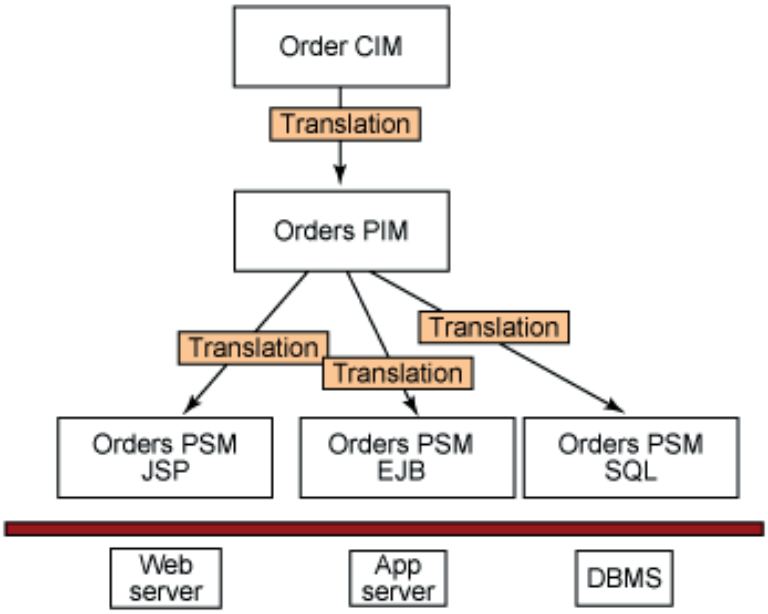


Figura 1: Modelos de MDA [7].

A pesar de las ventajas del desarrollo de software con MDE y considerando que dicho enfoque viene siendo utilizado durante los últimos por muchas personas, existen empresas que aún no lo utilizan. En [1] se señalados varios motivos, entre ellos, el desconocimiento de la tecnología MDE, de las herramientas y de los procesos utilizados para desarrollar aplicaciones.

Con finalidad de apoyar la iniciativa MDE, el presente trabajo pretende ser un instrumento de divulgación de esta tecnología al describir sus fundamentos teóricos, y señalar las principales actividades al construir varios artefactos MDE con algunas de las herramientas más difundidas en esta tecnología.

A continuación se señalan las diferentes secciones en las cuales se organiza el resto del artículo. La sección 2 describe la utilización de tecnología en la industria y los inconvenientes de su utilización. La sección 3 señala los fundamentos de MDE en base a DSLs. En la sección 4 se señalan y describen algunas de las herramientas para construir artefactos MDE. La sección 5 muestra un ejemplo de desarrollo de un DSL para manejar máquinas de estados básicas. En la sección 6 se hace referencia a las conclusiones y trabajos futuros.

2. Utilización de MDE en la Industria

A continuación se presentan varios trabajos de investigación que hacen referencia a la experiencia de utilización de MDE en la industria. También se señala algunos de los trabajos que se han enfocado en la descripción de la tecnología y herramientas utilizadas.

Kulkarni y Reddy [8] consideran que resulta complicado entender y aplicar la tecnología MDE en el desarrollo de software, además se señala que se necesita hacer una alta inversión en el desarrollo y/o adquisición de varios artefactos, como lenguajes adecuados, mecanismos de transformación de modelos, generadores de código.

Sánchez et al. [9] describe la utilización de MDE en dos empresas. Se considera que su adopción en la industria es limitada por factores de índole tecnológico y social.

Hutchinson et al. en [10] se señala la percepción de la utilización de MDE en la industria. Descubren varios casos de éxito de su utilización, así como también otros de fracaso. Se señalan las ventajas de utilización de MDE, entre ellos la mejora de la productibilidad de desarrollo, al automatizar el proceso de generación automática de código interoperabilidad. Sin embargo, también se señala factores por los cuales esta tecnología no es masivamente utilizada, entre ellos el desconocimiento de la tecnología y de las herramientas de desarrollo, la falta de un proceso formal de desarrollo, así como también se considera que muchas empresas no se arriesgan a hacer sus cosas de otra manera.

Mohagheghi et al. en [1] se señala varios motivos por los cuales MDE tiene dificultas para ser utilizada masivamente en las empresas, entre ellos: elevados costos en el entrenamiento del personal, así como también en el desarrollo y/o adquisición de varias herramientas de desarrollo de software, b) la presencia de riesgo al introducir un nuevo proceso de desarrollo, c) se considera que las herramientas utilizadas en MDE no son lo suficientemente maduras, d) desconocimiento de los entornos en los cuales MDE puede ser utilizado, e) desconocimiento de la tecnología.

Sánchez et al. en [9] se considera que la utilización de MDE en el desarrollo de software puede demorar algún tiempo más. Considera que se debe proveer mayor madurez en la herramientas de desarrollo, se debe contar con personal capacitado en la tecnología, y que se debe hacer conocer los beneficios de su utilización.

En cuanto a trabajos que describen la tecnología y herramientas utilizadas, se puede mencionar los siguientes.

Muhanna y Pick en [11] se centran en describir la tecnología de desarrollo de software en base a modelos, se hace hincapié en los niveles de abstracción que se logra con la tecnología, se explica lo que es un metamodelo dentro de un DSL y las herramientas computaciones que se tiene para su construcción y administración.

Kurtev et al. en [12] hace referencia a la evolución de la ingeniería de software. También se presenta un conjunto de prácticas y un conjunto de problemas que podrían ser resueltos con el enfoque MDE/DSL. En [13] Mernik et al. se muestra cómo y cuándo desarrollar lenguajes de dominio específico.

3. Fundamentos de MDE/DSL

El desarrollo de software con el enfoque MDE/DSL cubre dos fases, en la primera fase, ingenieros de software (expertos en modelado) deben desarrollar varios artefactos MDE, entre ellos, lenguajes en un dominio concreto (DSLs), mecanismos de transformación de modelos, y generadores automáticos de código. En la segunda fase se considera que personas no expertas en desarrollo de software (experto del dominio) puedan desarrollar aplicaciones utilizando los artefactos señalados en la primera fase [14].

Un modelo debe estar especificado a través de un DSL bien definido con sintaxis y semántica precisas. La sintaxis abstracta se define por lo general mediante un metamodelo, la sintaxis concreta mediante una representación textual o gráfica, mientras que su semántica viene relacionada con el significado de los modelos, es decir lo que se quiere representar con los ellos.

3.1. Sintaxis abstracta

Para definir la sintaxis abstracta de un DSL se tiene dos técnicas: metamodelado (modelware) y gramáticas libres de contexto (grammarware) [15]. En el caso de usar gramáticas se utilizan extensiones BNF (Backus-Naur Form). Mientras que el metamodelado considera el desarrollo de un metamodelo, el cual define los elementos

y reglas que especifican el modelo, a través de una especie de diagrama de clases. El meta-modelo define la gramática del lenguaje. Un modelo es a su metamodelo, de la misma manera que un programa de computación es a la gramática del lenguaje. Un metamodelo se representa mediante un lenguaje denominado *"lenguaje de metamodelado".* Para completar la especificación precisa de los modelos se utiliza restricciones OCL (Object Constraint Language) [16, 17].

3.2. Sintaxis concreta

Definida la sintaxis abstracta del DSL, se tiene que establecer su sintaxis concreta, la cual viene a ser el conjunto de símbolos textuales/gráficos que representan los elementos del modelo. Es lo que el usuario va a ver y utilizar. Es recomendable utilizar elementos textuales y gráficos con los cuales estén familiarizados los usuarios finales [18].

3.3. Transformación de modelos

MDE considera que se debe automatizar las tareas vinculadas al manejo de modelos en el desarrollo de software al utilizar este paradigma. Se deben contar con mecanismos que permitan, entre otros: depurarlos, validarlos, extenderlos. Sin duda, el principal reto de MDE, es automatizar las tareas señaladas, y la automatización no es otra cosa que desarrollar mecanismos de transformación entre modelos [19]. La transformación de modelos puede verse como la manipulación de modelos, para representar distintos niveles aspectos del software [20]. Las transformaciones de modelos pueden ser de dos tipos: Modelo a Modelo (M2M) y Modelo a Texto (M2T).

3.4. Generadores de código

La mejora de la productividad en el desarrollo de software con MDE se logra a través de la generación automática de código. El código generado puede ser en cualquiera de los lenguajes de programación conocidos, como Java, C++, PHP. Los programas pueden ser compilados para ser ejecutados en cualquier plataforma hardware. Para desarrollar generadores de código, existen varias herramientas, las cuales se ejecutan en base a una serie de reglas de trasformación y plantillas de generación.

4. Herramientas de desarrollo de MDE

La Tabla 1 presenta las principales arquitecturas y herramientas de desarrollo utilizadas en MDE.

Tabla 1: Arquitecturas y herramienta en MDE.

Arquitecturas de desarrollo	**Sintaxis abstracta Lenguaje de metamodelado**	
OMG	MOF	
Eclipse/EMF	Ecore/OCL	
Microsoft DSL Tools	Domain Model	
MetaCase/MetaEdit+	GOPRR	
Sintaxis concreta	**Transformación de modelos**	**Generación de código**
Lenguajes gráficos	QVT, ATL, RubyTL	Acceleo, Xtend
Sirius, GMF, EuGENia, Graphiti, Spray, Microsoft DSL Tools, MetaEdit+, Devil, AToM^3		
Lenguajes textuales		
Xtext, EMFText, TCS		

4.1. Arquitecturas de desarrollo

Arquitectura de OMG: MOF (http://www.omg.org/mof/) (Meta Object Facility) es el lenguaje de metamodelado utilizado por UML. MOF tiene una versión más pequeña denominada EMOF (Essential MOF). *URL: http://www.omg.org/mof/*

Arquitectura de Eclipse/EMF: EMF (Eclipse Modeling Framework) es el framework de Eclipse que permite trabajar con modelos en el desarrollo de software. Ecore es el lenguaje de metamodelado empleado en EMF [21]. Ecore es un subconjunto de MOF. Al igual que en UML, Ecore también utiliza OCL para especificar ciertas restricciones al modelo.

URL:http://www.eclipse.org/modeling/mft/search/concepts/subtopic.html

4.2. Sintaxis concreta (Editores gráficos)

Graphiti: utiliza GEF y Draw2D de Eclipse para crear editores altamente sofisticados. Es compatible con los EMF en el lado del dominio. Los diagramas son escritos por un metamodelo (Ecore) independiente de la plataforma y el diagrama de datos es separado de los datos de dominio. El API de Graphiti es estrictamente centrada en

el usuario y la creación de editores visuales se realiza en Java. *URL: https://eclipse.org/graphiti/*

Sirius: es un proyecto de Eclipse que permite crear fácilmente su propio workbeach de modelado gráfico mediante el aprovechamiento de las tecnologías de modelado de Eclipse, incluyendo EMF y GMF. Proporciona un workbeach genérico para la ingeniería basada en modelos. Puede ser fácilmente adaptado a cualquier necesidades específica sobre la base de enfoques de punto de vista *(viewpoint)*. Sirius se apoya en Acceleo, y otros proyectos, para facilitar el establecimiento de las relaciones entre los datos del modelo y su representación gráfica. *URL: https://wiki. eclipse.org/Sirius*

4.3. Sintaxis concreta (Editores textuales)

Xtext: este framework permite la creación de DSLs textuales. Es a través del metamodelo del DSL que se puede generar de manera automática una gramática del DSL. A partir esta gramática es posible representar modelos utilizando los elementos y relaciones expresadas en el metamodelo. *URL: https://eclipse.org/Xtext/*

EMFText: al igual que Xtext, EMFText permite definir una sintaxis textual a partir de un metamodelo Ecore. El código generado por EMFText es totalmente personalizable. Es posible realizar el análisis y validación completa de la sintaxis, lo que permite señalar los posibles errores que en ella puedan existir. *URL: http://www.emftext.org/index.php/EMFText*

4.4. Transformación de modelos

ATL: es un lenguaje de transformación de modelos y un conjunto de herramientas. En el campo de MDE. ATL proporciona formas de producir un conjunto de modelos objetivo a partir de un conjunto de modelos fuente. *URL: http://www.eclipse. org/atl/*

QVT: Query/View/Transformation es el estándar que la OMG propone para realizar transformaciones de modelo a modelo (OMG, 2008). Este metamodelo está compuesto por tres sublenguajes: QVT Operational Mappings, QVT Relations, y QVT Core. En este capítulo se da una visión general de QVT, para centrarse luego en QVT Operational Mappings [2]. *URL: http://www.omg.org/spec/QVT/*

4.5. Lenguajes de generación de código

Acceleo: es un proyecto de código abierto que tiene como objetivo proporcionar un generador de código (transformación de modelos a código) para las personas que pretenden sacar provecho de un enfoque basado en modelos para aumentar la productividad en el desarrollo de software [22]. Cuenta con licencia pública Eclipse Public License (EPL). Acceleo actualmente es considerado como una de las herramienta más potentes en la generación de código. *URL: https://www.eclipse.org/acceleo/*

Xtend: es un lenguaje de programación de tipo estático que se traduce en código fuente Java comprensible. Sintácticamente y semánticamente Xtend tiene sus raíces en el lenguaje de programación Java, pero mejora en muchos aspectos. Soporta M2M y M2T. Se utiliza como lenguaje "compañero" de Xtext. *URL: http://eclipse.org/xtend/*

5. Desarrollo de un DSL

En esta sección se describe el proceso de desarrollo de un DSL para el manejo de máquinas de estados básicas. Para ello primero, se diseña el metamodelo correspondiente que representa la sintaxis abstracta del DSL, utilizando la plataforma *EMF/Ecore*. Una vez definido y probado el metamodelo se debe crear su sintaxis concreta, para lo cual se desarrolla un editor textual utilizando *Xtext* y un editor gráfico utilizando *Sirius*. Finalmente se utiliza *Acceleo* para generación automática de código HTML a partir de un modelo de máquina de estados.

5.1. Requisitos del DSL

La máquina de estados debe permitir representar algunos tipos de estados: estado inicial, estado final, estados simples, estados compuestos que puedan contener estados simples; transiciones que unen los diferentes estados, así como también la posibilidad de representar diferentes recursos que pueden utilizar los estados.

5.2. Sintaxis abstracta (metamodelo)

En base a los requisitos establecidos en la sección anterior, se crea su metamodelo que puede ser representado en diferentes formatos, como diagrama de clases (Figura 2), XMI (XML Metadata), código java, árbol (tree). En este caso el metamodelo es representado Ecore. Para diseñar el metamodelo se selecciona el IDE Eclipse, dentro de

éste se utiliza el framework EMF. El proceso de diseño del metamodelo es el siguiente: 1) crear un proyecto EMF, para lo cual se selecciona la opción *Empty EMF Projec*, 2) poner un nombre al proyecto, 3) en la carpeta *model* se debe crear un modelo Ecore, 4) poner un nombre al metamodelo, 5) diseñar el metamodelo con los diferentes elementos constitutivos, como EClass, EReferences, EAttributes, etc. Para diseñar el metamodelo en forma gráfica se puede utilizar el utilitario *Ecore Diagram* de *Ecore Tools*.

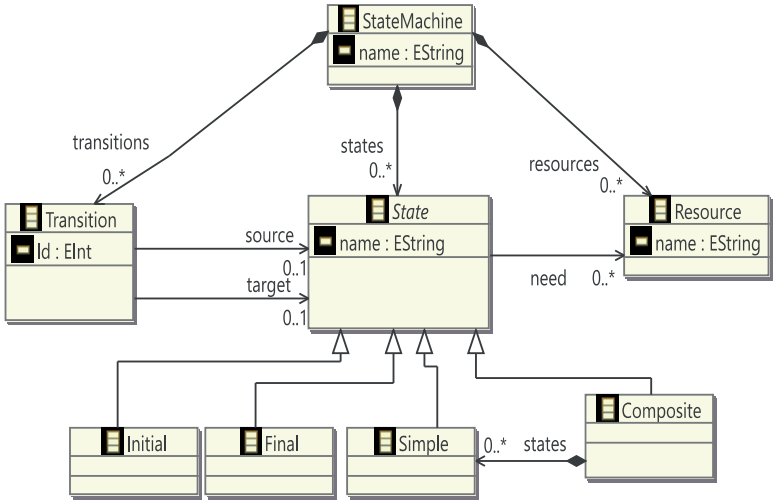


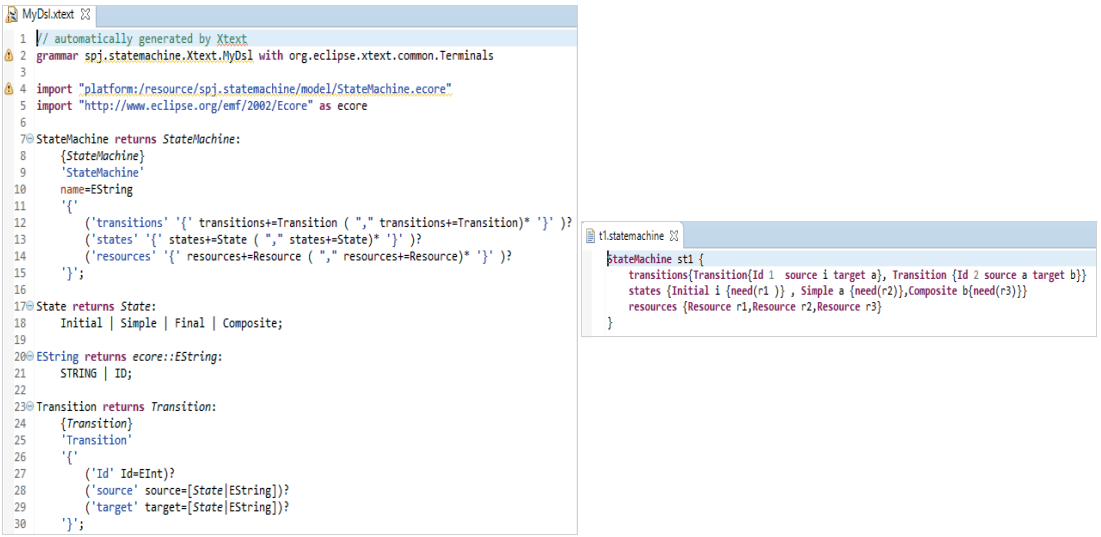
Figura 2: Metamodelo representado como diagrama de clases.

5.3. Sintaxis concreta (editor textuales)

Para desarrollar el lenguaje de programación para representar máquinas de estados se utilizó Xtext, debido a que ofrece un lenguaje específico para diseñar gramáticas tipo BNF. La Figura 3a muestra un extracto de la gramática generada automáticamente por Xtext, la cual es derivada del metamodelo (Figura 2). La Figura 3b muestra el editor textual que permite escribir instrucciones en el nuevo lenguaje de propósito específico. El código fuente del programa se registra en archivos con extensión (*.statemachine).

El proceso de creación de un proyecto Xtext, considera las siguientes actividades:

1) creación de un proyecto Xtext a partir de un metamodelo existente, 2) especificar el nombre del lenguaje, 3) especificar el nombre de la extensión del lenguaje, 4) modificar la gramática creada automáticamente si así se lo requiere, 5) crear los artefactos Xtext a partir de la gramática, 6) ejecutar el proceso MWE2 Workflow, 7) utilizar el editor textual para crear programas con el nuevo lenguaje de propósito específico.



(a) (b)

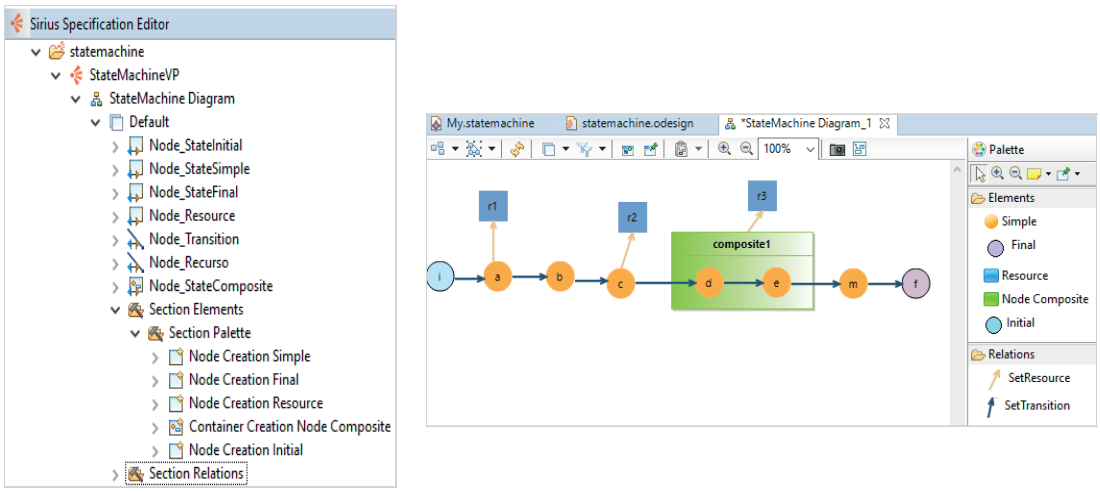
Figura 3: (a) Extracto de la gramática del lenguaje, (b) Editor textual.

5.4. Sintaxis concreta (editor gráfico)

Tomando como base el metamodelo de la máquina de estados, se diseña la forma gráfica que tendrá cada uno de elementos del metamodelo. Se utilizó Sirius debido a que permite la especificación breve y simple de un workbench de modelado en términos de "*vistas*" y que pueden tener la forma de editores gráficos, tablas o árboles, con reglas de validación y acciones usando descripciones declarativas. Todas las características y comportamientos de los vistas se pueden configurar fácilmente con un conocimiento técnico mínimo.

La Figura 4a muestra el entorno de configuración de una vista y la Figura 4b el entorno para representar máquinas de estados de forma gráfica. Cabe señalar que cada modelo de una máquina de estados es una *instancia* del metamodelo. Sirius utiliza AQL (Acceleo Query Language), como lenguaje recomendado (a partir de Sirius 3.1) para acceder a los elementos del metamodelo (realización de consultas).

El proceso de creación de un proyecto Sirius, considera las siguientes actividades: 1) creación de un proyecto Viewpoint Specification Project, 2) especificar un Viewpoint, 3) especificar el tipo de Representación (Diagram, Edition Table, Cross Table, Tree, Sequence Diagram), 3) mapeo entre los elementos gráficos del diagrama con los elementos del metamodelo, 4) especificar las acciones que se van a realizar cuando se produzca un evento, y 5) especificar los elementos de la barra de herramientas del editor (paleta).



(a) (b)

Figura 4: (a) Especificación del editor, (b) Entorno gráfico.

5.5. Generador de código

Para desarrollar un generador de código (M2T) se utilizó Acceleo, debido a su facilidad de uso y a que ofrece ventajas como alta capacidad de personalización, interoperabilidad, lanzamiento fácil para generar un programa en cualquier lenguaje de programación. En la Figura 5a se señala la configuración del template para generar código a partir de un modelo. La Figura 5b muestra el código HTML generado.

El proceso de creación de un proyecto Acceleo considera las siguientes actividades: 1) creación de un proyecto Acceleo, 2) seleccionar el metamodelo a partir del cual generar el proyecto, 3) configuración de la plantilla generate.mtl para tomar que toma los datos de un modelo (instancia del metamodelo), 4) ejecutar una aplicación Acceleo a partir de la plantilla generate.mtl, se debe especificar el modelo de entrada y la ubicación del archivo generado (código fuente generado a partir del modelo).

6. Conclusiones y trabajos futuros

Desde su aparición los lenguajes de programación han ido evolucionando con la finalidad de hacer cada vez más fácil y rápida la tarea de desarrollar software. Desde hace algunos años han aparecido varias propuestas, una de ellas MDE que posibilita el poder desarrollar software a través de modelos expresados a través de DSLs. Como postulados principales de esta tecnología se considera el mejorar la productividad y calidad del proceso de desarrollo. Sin embargo, a pesar de ello, son varios los obstáculos que se deben superar, para que MDE sea ampliamente utilizado por la comunidad de

```
khtml>
<body>
    <ul>
    <h3>Elementos de la Maquina de Estado</h3>
    <h4>Transiciones:</h4>
    [for (t:Transition|aStateMachine.transitions)]
        [file('stdout',false)]procesando un envio:[/file]
        <li>Transicion ([t.Id /]) une el estado ([t.source.name/])
        con el estado ([t.target.name/])</li>
    [/for]
    <h4>Estados:</h4>
    [for (s:State|aStateMachine.states)]
        [file('stdout',false)][/file]
        <li>Estado ([s.name /]) con recurso ([s.need.name/])</li>
    [/for]
    <h4>Estados compuestos:</h4>
    [for (c:Composite|aStateMachine.states)]
        [file('stdout',false)][/file]
        <li>Estado ([c.name/]) con sub-estados:</li>
        [for (s:State|c.states)]
            [file('stdout',false)][/file]
            <li>([s.name/])</li>
        [/for]
    [/for]
    </ul>
</body>
</html>
[/file]
[/template]
```

(a)

Elementos de la Maquina de Estado

Transiciones:

- Transicion (1) une el estado (i) con el estado (a)
- Transicion (2) une el estado (a) con el estado (b)
- Transicion (3) une el estado (b) con el estado (c)
- Transicion (4) une el estado (c) con el estado (d)
- Transicion (5) une el estado (d) con el estado (e)
- Transicion (6) une el estado (e) con el estado (m)
- Transicion (7) une el estado (m) con el estado (f)

Estados:

- Estado (i) con recurso ()
- Estado (a) con recurso (r1)
- Estado (b) con recurso (r2)
- Estado (composite1) con recurso (r3)
- Estado (c) con recurso ()
- Estado (f) con recurso ()
- Estado (m) con recurso ()

Estados compuestos:

- Estado (composite1) con sub-estados:
- (d)
- (e)

(b)

Figura 5: (a) Templete que genera código, (b) Código HTML.

desarrollo. Por tal motivo el presente trabajo pretende aportar con la iniciativa de MDE, a través de la explicación rápida y precisa de los fundamentos de esta tecnología, la descripción de varias herramientas, así como también la descripción de las actividades que se tienen que realizar en el proceso de desarrollo de varios artefactos MDE.

Muchos son los trabajos de investigaciones que se han realizado en torno a MDE, cada uno abordando un aspecto en particular, sin embargo este trabajo logra organizar y explicar los elementos más relevantes de la tecnología MDE. Se han identificado varias áreas que en un futuro pueden ser desarrolladas y mejoradas, entre ellas: la adopción de metodologías (procesos formales de desarrollo de software) con este enfoque, el desarrollo de aplicaciones MDE por el experto del dominio (usuario final).

Referencias

[1] Mohagheghi, P., et al., *Where does model-driven engineering help? Experiences from three industrial cases*. Software & Systems Modeling, 2013. 12(3): p. 619-639.

[2] García, J., et al., *Desarrollo de Software Dirigido por Modelos: Conceptos, Métodos y Herramientas*. 2013. 586.

[3] Mohagheghi, P. and V. Dehlen. *Where is the proof?-A review of experiences from applying MDE in industry. in European Conference on Model Driven Architecture-Foundations and Applications.* 2008. Springer.

[4] Schmidt, D.C., *Model-driven engineering. COMPUTER-IEEE COMPUTER SOCIETY-*, 2006. 39(2): p. 25.

[5] Kent, S. *Model driven engineering. in International Conference on Integrated Formal Methods*. 2002. Springer.

[6] Dänekas, C., et al., *Towards a model-driven-architecture process for smart grid projects. Digital enterprise design & management*, 2014. 261: p. 47-58.

[7] Pérez, J., F. Ruiz, and M. Piattini, *Model driven engineering aplicado a business process management*. Informe Técnico UCLM-TSI-002, 2007.

[8] Kulkarni, V. and S. Reddy. *Model-driven development of enterprise applications. in International Conference on the Unified Modeling Language*. 2004. Springer.

[9] Cuadrado, J.S., J.L.C. Izquierdo, and J.G. Molina, *Applying model-driven engineering in small software enterprises. Science of Computer Programming*, 2014. 89: p. 176-198.

[10] Hutchinson, J., et al. *Empirical assessment of MDE in industry. in Software Engineering (ICSE)*, 2011 33rd International Conference on. 2011. IEEE.

[11] Muhanna, W.A. and R.A. Pick, *Meta-modeling concepts and tools for model management: a systems approach. Management Science*, 1994. 40(9): p. 1093-1123.

[12] Kurtev, I., et al. *Model-based DSL frameworks. in Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006. ACM.

[13] Mernik, M., J. Heering, and A.M. Sloane, *When and how to develop domain-specific languages. ACM computing surveys (CSUR)*, 2005. 37(4): p. 316-344.

[14] Jácome Guerrero, S., *Descripción de las actividades de una propuesta de Metodología de Desarrollo de Software Dirigida por Modelos*, 2013.

[15] Wimmer, M. and G. Kramler. *Bridging grammarware and modelware. in International Conference on Model Driven Engineering Languages and Systems*. 2005. Springer.

[16] Richters, M. and M. Gogolla. *On formalizing the UML object constraint language OCL. in International Conference on Conceptual Modeling*. 1998. Springer.

[17] Richters, M., *A precise approach to validating UML models and OCL constraints* 2002: Logos.

[18] Izquierdo, J.L.C., et al. *Engaging end-users in the collaborative development of domain-specific modelling languages. in International Conference on Cooperative Design, Visualization and Engineering*. 2013. Springer.

[19] Czarnecki, K. and S. Helsen, *Feature-based survey of model transformation approaches*. IBM Systems Journal, 2006. 45(3): p. 621-645.

[20] Montenegro Marín, C.E., et al., *Application of model-driven engineering (MDA) for the construction of a tool for domain-specific modeling (DSM) and the creation of modules in learning management systems (LMS) platform independent*. Dyna, 2011. 78(169): p. 43-52.

[21] Haan, J., *The Entrerprise Architect: Building an Agile Enterprise*. Blog para divulgaçao de pesquisas e soluçoes tecnológicas que envolvem MDE, MDA e DSL. Disponıvel em: http://www.theenterprisearchitect. eu/Acesso em, 2010. 21.

[22] Kara, I.B., *Design and Implementation of the ModelicaML Code Generator Using Acceleo 3*. X, 2015.