Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 4, No. 1, pp. 22-33, Junio 2017
Fecha recepción: 28/04/2017, Fecha aceptación: 09/08/2017

Hacia un Enfoque para la Generación de Código con MDA

Mercedes Picón[1], Javier Torrealba[2]
mpicon@una.edu.ve, jtorrealba@una.edu.ve

[1] Centro Local Metropolitano-Ingeniería de Sistemas, Universidad Nacional Abierta, Caracas, Venezuela
[2] Coordinación de Ingeniería de Sistemas, Universidad Nacional Abierta, Caracas, Venezuela

Resumen: Se presentan opciones disponibles y posibles riesgos a considerar al seleccionar herramientas y métodos para generar código con MDA. Con el objetivo de complementar la información suministrada por los proveedores de herramientas MDA, se propone un enfoque que guíe la selección de generadores automáticos de código. Para reducir el riesgo que suponen los procesos pesados asociados con frecuencia a MDA, se incluyeron prácticas Ágiles en el enfoque propuesto. Muchas herramientas apoyan la generación de código con base en modelos UML y las herramientas seleccionadas van a influir en todas las fases de los proyectos de desarrollo y en el mantenimiento de sistemas. Sin embargo, las referencias citadas afirman que se trata de una tecnología en evolución y que está por alcanzar su madurez.

Palabras Clave: Generadores de Código; UML; MDA; Selección de Herramientas MDA; Prácticas Agile; Codificación.

Abstract: Available options and potential risks to consider are presented regarding the selection of tools and methods to generate code with MDA. To supplement information provided by MDA tool vendors, an approach is proposed to guide the selection of automatic code generators. To reduce risks frequently related to heavy-weight MDA processes, Agile practices have been included in the proposed approach. Many tools support UML based code generation and the selected tools will impact all project development phases and system's maintenance. However, the cited references claim it's an evolving technology which has yet to reach maturity.

Keywords: Code Generators; UML; MDA; MDA Tool Selection; Agile Practices; Coding.

I. Introducción

Mediante prácticas estándar de programación es posible producir en forma automática código completo y correcto basado en diagramas UML (Unified Modeling Language) o Lenguaje Unificado de Modelado [1]. A partir de diagramas de estado, se ha producido código para componentes de comportamiento de alta complejidad en sistemas de tiempo real y embebidos [1]. Sin embargo, aún mediante herramientas que apoyan la MDA (Model Driven Architecture) o Arquitectura Guiada por Modelos, particularmente aquellas que transforman modelos en código, no logra producirse la totalidad del código para algunos componentes de software [2][3][4].

Los métodos y herramientas de MDA para generar código en forma automatizada están evolucionando hacia su madurez [3][5], pero todavía se carece de metodologías con procesos que puedan ser instanciados para apoyar MDA y en particular, para orientar la selección de generadores de código basados en UML (GC) [4][6]. Con el propósito de diseñar un enfoque que pueda guiar la generación de código mediante GC, se investigaron prácticas de la MDA y su integración con metodologías Ágiles.

A partir de esta introducción, el presente artículo está organizado en las siguientes secciones: en la Sección II se describe brevemente el enfoque MDA; la Sección III presenta prácticas con modelos UML para la generación automática de código; en la Sección IV se describen características de las herramientas MDA para la generación automática de código. Una clasificación de posibles riesgos en los procesos anteriores se ofrece en la Sección V. La Sección VI propone un enfoque para emplear MDA, conjuntamente con prácticas ágiles en la generación automática de código y finalmente, la Sección VII presenta las conclusiones que resultaron de esta investigación.

II. El Enfoque MDA

La MDA plantea la obtención de valor de los modelos y del modelado, como apoyo al manejo de la complejidad e interdependencia de sistemas complejos [7]. En 2001 el OMG presentó la MDA como un enfoque para la MDE (Model Driven Engineering) o Ingeniería Guiada por Modelos. MDE es un enfoque que surgió para incrementar el nivel de abstracción en la especificación de programas y la automatización en el desarrollo de programas [8]. Los estándares de la MDA incluyen los siguientes [7]:

- XMI (XML Metadata Interchange) como mecanismo estándar para intercambio,
- UML como lenguaje para modelado mediante diagramas,
- CWM (Common Warehouse Metamodel) como estándar para almacenes de datos y
- MOF (Meta Object Facility) como mecanismo para analizar modelos en XMI, el cual provee los constructos estándar para modelado e intercambio en la MDA.

UML está conformado por una familia de lenguajes para especificar, visualizar, construir y documentar el diseño y la estructura de sistemas de software [9][10]. UML 2.0 comprende diagramas dentro de las categorías de estructura, de comportamiento y de interacción [9]. En el análisis y diseño bajo el enfoque OO (Orientado a Objetos), se desarrollan diagramas UML hasta obtener un formato utilizable por los programadores [11].

MDA define los siguientes niveles de modelado con UML [7]:

- CIM (Computation Independent Model) o Modelo Independiente de Cómputo, se emplea para representar los modelos de negocio y requerimientos de sistemas [12]
- PIM (Platform Independent Model) o Modelo Independiente de Plataforma, es generado sin detalles sobre las soluciones técnicas y es convertible en uno o más modelos de plataforma específica [13]
- PSM (Platform Specific Model) o Modelo de Plataforma Específica, puede convertirse en el código diseñado para una plataforma específica [13]. Se produce a partir del PIM mediante herramientas MDA [4]. Al cambiar los requerimientos del negocio se modifica el PIM en vez los modelos PSM, los cuales se regeneran a partir del PIM modificado [14].

Al separar los modelos PIM de la información técnica relacionada con plataformas específicas, la MDA incrementa la reusabilidad de PIMs y PSMs [15]. Los modelos PIM y PSM son de interés particular en la generación automática de código basada en UML. No obstante, no hay un método estándar para construir modelos CIM ni para transformarlos en uno o varios PIM en niveles más bajos de abstracción, siendo el PIM el objetivo final de la construcción de modelos que permitan la generación de código [12][14].

El MDD (Model Driven Development) o Desarrollo basado en Modelos es un subconjunto de MDE que se concentra en transformaciones que generan representaciones más concretas a partir de abstracciones [13][16][17]. Plantea desarrollar modelos extensos antes de escribir código fuente. Tanto MDD como MDA pueden categorizarse como sub-secciones de MDE [17].

Un estudio de seis metodologías basadas en MDA concluyó que no ofrecen un proceso de desarrollo que se pueda instanciar [4]. La selección de GC no es apoyada en las metodologías analizadas en ese estudio. Se desarrolló una metodología basada en MDA que se considera posible de instanciar [4]. Sin embargo, en su actividad para selección de herramientas solo recomienda determinar cuáles son los requerimientos para estas herramientas MDA y compararlos con las capacidades de herramientas disponibles en el mercado.

III. Prácticas Basadas en Modelos UML para la Generación Automática de Código

Se describen prácticas basadas tanto en enfoques prescriptivos como en metodologías ágiles. Estas prácticas han sido sugeridas por desarrolladores de métodos y herramientas MDA, por proyectos que emplearon GC y por estudios acerca de métodos y herramientas. Se discuten en esta sección porque son necesarias para la codificación automatizada y para ajustar los modelos durante iteraciones que conduzcan a elevados porcentajes de código correcto.

MDA presenta pocos detalles a seguir para las transformaciones de un nivel de modelado a otro hasta obtener el código [2]. Los equipos que han desarrollado herramientas que transforman modelos a modelos o a código, así como también los proyectos que emplearon GC, proveen prácticas y recomendaciones a considerar en el desarrollo y mantenimiento de sistemas con MDA. Se presentan en primer lugar las pertinentes al modelado UML y en segundo lugar, las relacionadas con los dos tipos de enfoques de MDA.

A. Prácticas de Modelado

El diagrama de clase es el punto inicial para generar la estructura básica del código. Posteriormente, el cuerpo de los métodos se complementa con diagramas de secuencia y diagramas de máquinas de estado [5]. Los diagramas de casos de uso son empleados principalmente en modelos CIM. PIM utiliza todos los diagramas UML excepto los de componente y de despliegue. PSM tiene diagramas de componente y de clase. Algunos proyectos expresan PSM como formatos estándar de código y comentarios significativos, lo que es aceptado por la perspectiva y los principios ágiles [13].

La propuesta de Agile UP describe lo que puede considerarse una ampliación de los diagramas UML asociados a la Arquitectura de las 4+1 vistas, para distintos objetivos de modelado [18][19]. La Tabla I muestra solo los diagramas UML correspondientes a los tipos de modelo identificados [18].

Tabla I: Tipos de Modelado y sus Diagramas UML

Modelado	Diagramas UML utilizados en tipos de Modelado
de Utilización	Casos de Uso
Conceptual de Dominio	Clases
de Procesos	Actividad
Arquitectónico	Componentes, Despliegue, Paquetes
de Objetos Dinámicos	Comunicación, Estructura Compuesta, Resumen de Interacción, Secuencia, Máquina de Estado, Temporización
Estructural detallado	Clases, Objeto

Los modelos proveen un nivel de abstracción que facilita la representación de la información arquitectónica y de diseño, así como la corrección de fallas de arquitectura y de diseño, la eliminación de los aspectos estructurales y funcionales innecesarios y la adaptación de la estructura para lograr funcionalidad adicional y optimización del comportamiento [10]. Sin embargo, un solo modelo es insuficiente para desarrollar un sistema de software completo, se requieren varios modelos en distintos lenguajes de modelado, e incluso código en lenguajes de programación [2].

Se pueden necesitar diagramas adicionales a los presentados la Tabla I [18][19], para evitar que el modelado con UML resulte en una representación incompleta de los componentes y de su código. Por ejemplo, para el modelado de la interfaz usuaria y el de requerimientos suplementarios, se recomiendan diagramas distintos a los de UML [18].

B. Prácticas que Combinan MDA y Metodologías Ágiles

Entre las iniciativas para integrar los enfoques de MDA con las propuestas de modelos ágiles están:

- Agile MDA, basada en la noción de que el código y los modelos ejecutables son operativamente lo mismo y, por lo tanto, los principios ágiles aplican a los modelos [17].
- AMDD (Agile MDD) promueve el uso de herramientas de apoyo simples y diagramas UML para modelar requerimientos con un proceso iterativo e incremental [20].

El enfoque de transformación se utiliza para producir modelos o artefactos de una tecnología específica; el de modelos ejecutables, recomendado para prácticas ágiles [10], se implementa en una plataforma tecnológica que ejecuta el modelo directamente, tratándolo como código fuente interpretable. Ambos enfoques logran la generación de código ejecutable a partir de modelos [7].

En el enfoque Ágil MDA, los modelos deben ser suficientemente completos para que puedan ser ejecutados por sí solos sin que sea necesario distinguir entre modelos a nivel de análisis y de diseño. En vez de transformar modelos, estos se conectan entre sí y se mapean hacia un único modelo combinado que, por último, es traducido al código según un sistema único de arquitectura MDA [13].

Mediante UML ejecutable (xUML), MDA ha logrado mejoras en la calidad del software al habilitar modelos UML para que puedan traducirse en código ejecutable [10]. Un modelo ejecutable UML puede tratarse como un PIM, de acuerdo con los conceptos de MDA [15].

Aunque xUML tiene limitaciones, se ha propuesto combinar este enfoque con prácticas ágiles, particularmente las que generan el código de pruebas automáticamente. Se ha considerado adecuado para sistemas embebidos en dispositivos dentro de los cuales en software no puede fallar [10]. Esto se ha justificado indicando que las prácticas ágiles como son XP y ASD, evitan separar las actividades de diseño de las de implementación y abandonan las actividades de documentación, para favorecer el desarrollo de pruebas rigurosas imprescindibles en sistemas de alta criticidad [10].

Existe una falta de compatibilidad de principios entre XP y MDA; la primera enfatiza aspectos medulares de la tecnología de software sin basarse en diagramas para documentar un sistema. Estudios que examinaron la integración de XP y el modelado con UML, recomiendan el Modelado Extremo como solución [21].

La forma frecuente de combinar MDA con metodologías ágiles es incorporar al enfoque MDA prácticas que incluyen XP (Extreme Programming), Scrum, DSDM (Dynamic Systems Development Method), ASD (Adaptive Software Development) y Crystal [17]. Las tres primeras son las más adoptadas en desarrollos de sistemas basados en modelos [17].

De una revisión realizada por Hansson y Zao, que abarcó 24 artículos de IEEE, 27 de ACM y 27 de Springerlink [17], se presenta a continuación una relación entre las prácticas ágiles y MDA, basada en los trabajos de Zhang y Patels:

- Modelado como codificación: UML como lenguaje de programación visual que es compilado por el GC ha generado C o C++
- Modelado iterativo e incremental: Se modela en UML durante múltiples sprints, repitiendo el mismo conjunto de actividades en cada sprint. Un modelo ejecutable evoluciona en cada sprint con un incremento funcional con respecto al modelo del sprint anterior. Los modelos se mantienen ejecutables tanto para la simulación como para pruebas en la plataforma de ejecución
- Modelado continuo: Corresponde a la práctica ágil de la integración continua que unifica los diseños en UML frecuentemente, con la generación automática de código para varias simulaciones en un sprint; se produce también en forma automática el código para pruebas dos o más veces en un sprint
- Codificación: generación de código C/C++ completamente ejecutable basado en diagramas de máquinas de estado orientadas a transiciones.

Con los mismos diagramas UML se puede generar el código fuente y el de pruebas. Esto aumenta la disponibilidad de las pruebas al inicio de las actividades de codificación. Con el enfoque "probar primero" los casos de prueba pueden basarse en diagramas de secuencia. La generación automática de código de programas y del código de pruebas, contribuye a identificar errores en los modelos [10].

Es necesario definir métodos que proporcionen pautas a seguir y técnicas a utilizar para desarrollar software con MDA [20]. Puede integrarse MDA con metodologías robustas como Rational Unified Process (RUP) o livianas como las propuestas ágiles, de forma que MDA provea los estándares y mecanismos para concebir una arquitectura alrededor de modelos y su proceso de transformación, y que las metodologías suministren lineamientos para la gestión del proyecto [20].

El uso de prácticas ágiles en el desarrollo de sistemas de tiempo real tendría que estar a cargo de expertos tanto en el dominio del sistema como en los métodos ágiles. Sin embargo, se considera que el rigor y robustez que aportan los modelos permite que MDA combinado con Agile pueda ser la base para desarrollar componentes de alta criticidad [17].

Las prácticas descritas anteriormente deben considerarse al fijar los criterios de selección de GC.

IV. Características de las Herramientas MDA para Generar Código con Base en Modelos UML

Se describen a continuación las características que deberían considerarse al seleccionar generadores de código basados en modelos UML (GC). Un GC es una herramienta que transforma un modelo conformado por diagramas UML en código fuente en lenguajes de alto nivel [1]. El modelo puede

ser un PSM que describa la estructura, el comportamiento o la arquitectura de los sistemas a desarrollar [22].

Estas herramientas incluyen ArgoUML, Rational Rhapsody, StarUML, Altova UModel, y Acceleo [23][24][25][26][27], entre otras. Sus características, extraídas principalmente de los manuales de estas herramientas, sirven para establecer los criterios de selección.

Entre las características de funcionalidad (idoneidad funcional) se encuentran las presentadas a continuación:

- Generación de código a nivel de producción en lenguajes: C, C++, Java, C#, Visual Basic, Perl, Python, PHP, entre otros.
- Generación de código para pruebas unitarias, de integración y de sistema [10].
- Diagramas UML procesados para generar código.
- Versión de UML: 2.4, 2.1, 2, 1.3.
- "Ida y vuelta" o "round-trip" para crear y ajustar diagramas a partir del código.
- Enfoque de generación de código: estructural, de comportamiento o de intercambio [1][28].
- Generación de código para definir y manipular datos en DBMS específicos.
- Integración con verificadores de modelos como SPIN [1].
- Capacidades de DSL (Domain Specific Language) o lenguajes de dominio específico.
- Transformaciones modelo a modelo CIM-PIM-PSM.
- Capacidades para diferenciación, comparación y refactorización de modelos, revisión de consistencia de modelos (Ej. SPIN), gestión de versiones e integración de modelos y co-evolución de modelos [2].
- Conformidad con estándares MDA.
- Uso especializado o de propósito general.

Entre las características de compatibilidad (coexistencia e interoperabilidad), están las siguientes relacionadas con los ambientes de desarrollo y operaciones:

- Plataforma Operativa: Linux, Mac OS X, MS Windows.
- Formato de archivo para intercambio de información: OCL (Object Constraint Language), XMI (XML Metadata Interchange), .uml (de StarUML).
- DBMS.
- Integración con ambientes y herramientas de desarrollo: Visual SourceSafe, CVS, MS Word, Eclipse, Visual Studio.NET.
- Integración con herramientas para transformación de modelos y con herramientas de pruebas de software, entre otros procesos de desarrollo.

Otras características de los GC incluyen:

- Proveedor: código abierto o comercial.
- Continuidad de la herramienta y proveedor
- Estabilidad de la versión.
- Servicios de Soporte técnico del proveedor.

Se detallan a continuación algunas de las características anteriores con recomendaciones para evaluar las herramientas:

- Diagramas UML procesados para generar código. Algunos GC procesan hasta 14 diagramas UML 2.4 [25]; otros no procesan diagramas de secuencia, de objeto, paquetes, temporización, ni de entorno de interacción; varios GC solo apoyan aspectos fundamentales de las máquinas de estado según la especificación UML 2.0 [15].
- Existen GC que producen código fuente genérico para aplicaciones que van a operar en una variedad de plataformas [29].
- Existen GC que pueden mapear entre distintas fuentes y destinos [25].
- "Ida y vuelta" o "round-trip" para crear y ajustar diagramas a partir del código [25]. Combinar la ingeniería hacia adelante y la ingeniería en reverso para obtener diagramas UML a partir del código y así documentar componentes de los sistemas, generar los diagramas UML mediante ingeniería en reverso y obtener PSM reutilizables.
- Los DSLs incluyen Domain Specific Modeling Languages (DSML) o lenguajes para modelado de dominio específico, los cuales se construyen con lenguajes meta-modelo de propósito general como MOF [30].
- Transformaciones Modelo a Modelo. Hay herramientas que facilitan la creación de un modelo PIM y su transformación a un modelo PSM. También transforman el modelo PSM en un código totalmente ejecutable en varios lenguajes de programación [2]. Por el contrario, hay transformaciones de PIM a PSM realizadas con lenguajes o plantillas propietarias, sin conformidad con OMG MDA. Estas transformaciones pueden además producir código incompleto [31].
- Para facilitar estas transformaciones algunos GC mejoran o corrigen los diagramas [1][28].
- Uso especializado o de propósito general. Los GC disponibles en la actualidad se han empleado en el dominio automotriz, en sistemas embebidos y de tiempo real, con requerimientos exigentes de confiabilidad. Sin embargo, UML no cubre las necesidades de modelado de los distintos dominios, por lo que pueden necesitarse GC especializados u otros métodos para generar código [10].
- Conformidad con estándares MDA. Incluye conformidad con subconjuntos de UML como fUML para la generación de código mediante modelos ejecutables [8].
- Plataforma Operativa y Sistema manejador de BD. Aunque el código generado tenga dependencias con ciertos sistemas de bases de datos y sistemas operativos, varias herramientas generan código ejecutable sobre una variedad de plataformas [29], incluyendo plataformas móviles. Existen GC que permiten generar código ajustado a un DBMS [25][26].
- Integración con ambientes y herramientas de desarrollo: Los métodos y herramientas GC deben integrarse con las plataformas de desarrollo y operacional. Los GC pueden ser componentes de Integrated Development Environments (IDE) o ambientes integrados de desarrollo que tienen conformidad con MDA.

Entre las funciones que ofrecen los GC de carácter experimental o académico están las siguientes:

- Generación de un PSM que representa N-Capas para Web con enlaces de trazabilidad entre los componentes de meta-modelos PIM y PSM y reglas de transformación con el lenguaje Atlas Transformation Language (ATL) [16].
- Transformación de máquinas de estado en UML 2.0 a C#; considerando todos los conceptos de estos diagramas y apoyando el desarrollo eficiente de aplicaciones multi-hilos, bien documentadas y con capacidad de mantenimiento [15].
- Generación del código C++ completo de un reloj digital y un juego de Tetris con base en diagramas de máquinas de estado [1].
- Producción de código Java a partir de diagramas de secuencia creados con un plug-in de Eclipse como IDE, para modelado con UML [32].

Un estudio realizado en el 2015 concluye que ninguna herramienta por sí sola puede apoyar las distintas actividades de MDA [2]. Entre estas actividades que complementan la generación de código destacan [2]:

- la interacción entre el modelo y el código así como entre modelos;
- transformaciones modelo a código, modelo a modelo, y código a modelo;
- funcionalidad para la cual se cuenta con menos apoyo como la integración de modelos, mapeo entre modelos, y fusión de modelos.

Las actividades anteriores serán de utilidad en las iteraciones de la fase de construcción para ajustar los modelos de forma que se eleve el porcentaje de código completo y correcto generado.

V. Riesgos en la Generación Automática de Código con Modelos UML

Los siguientes riesgos se pueden presentar en los procesos de MDA relacionados con la generación automatizada de código. Estos se clasifican aquí en dos categorías, según estén más relacionados con los métodos o con las herramientas:

A. Riesgos Asociados a MDA o UML

1) Elaboración de los Modelos: Riesgos relacionados con los diagramas que conforman los modelos y con su elaboración.

Los diagramas pueden estar elaborados en exceso o en forma insuficiente. Al elaborar los diagramas UML durante el modelado pueden omitirse atributos indispensables para generar código y los diagramas tendrían que completarse para ser procesados por los GC. Esto se debe en parte al énfasis de los diagramas en la comunicación usuario-analista.

Los diagramas creados en las iteraciones de los flujos de trabajo de requisitos, análisis y diseño, intentan comunicar modelos con diferente nivel de detalle, entre usuarios y desarrolladores [11]. Estos diagramas pueden estar insuficientemente elaborados para generar código; pueden omitir elementos indispensables para codificar, ej. el recuadro que indica bucles o repeticiones en un diagrama de secuencia.

Al trasformar CIMs a PIMs o PIMs a PSMs, la pérdida de información puede impedir la generación de código. También es posible que deban elaborarse más los modelos para llegar al PIM con los diagramas y nivel de elaboración adecuados para generar código completo. Un modelo incompleto puede incluir diagramas UML sin suficiente elaboración para permitir la programación manual o automática.

2) MDA: Este enfoque presenta los siguientes riesgos posibles en relación con el uso de GC.

La modalidad con UML ejecutable propone que el sistema se especifique en su totalidad dentro del PIM. Este enfoque propuesto por Shlaer-Mellor afirma que a partir del PIM tanto el PSM como el código son generados en un 100% [33]. Sin embargo, la existencia de dos modalidades para implementar MDA, plantea la posibilidad de una elección inapropiada [33].

En la otra modalidad hay enfoques de transformaciones de modelos que implementan en forma inadecuada las actuales especificaciones de UML. Un ejemplo es la automatización insuficiente para adaptar los diagramas de secuencia al modificar los de estado y para ajustar los de objeto al cambiar los diagramas de clase correspondientes. Esto influye en la transformación automática de modelos a código correcto [10] y favorece la selección de la primera modalidad de MDA.

MDA cambia el énfasis del desarrollo desde los programas hacia los modelos [29]. Su integración con las prácticas ágiles debe considerar que estas no promueven el modelado, ni la construcción de dos modelos, uno de análisis y otro de diseño. Aunque se afirma que combinar MDA y prácticas ágiles reduce las limitaciones de ambos enfoques [17], los desarrolladores tendrían contar con experiencia previa en esta integración para reducir el riesgo del proyecto.

MDA plantea que con PIM se crea el PSM en forma automática, entonces se trata de tener un PIM completo teniendo en cuenta que MDA no prescribe ningún proceso de desarrollo específico para desarrollarlo. Cada proyecto de desarrollo con MDA debe definir su propio proceso de desarrollo o seleccionar un proceso dentro de un conjunto muy pequeño de metodologías MDA disponibles [34].

3) UML: Riesgos relacionados con UML como lenguaje de modelado.

El diseño de lenguajes de modelado posee una base teórica reducida [1]. Carece también de un mapeo completo entre diagramas UML y código [35]. Estas limitaciones pueden permitir inconsistencias al transformar los diagramas a código como en el caso de no generar asociaciones que pueden ser necesarias entre clases [28]. La limitación anterior puede dificultar el establecimiento de relaciones precisas entre diagramas UML y tipos de componente como la interfaz usuaria, el manejo de base de datos y la lógica empresarial.

Acerca de la relación entre la lógica de comportamiento y la capa de lógica empresarial, es notoria la falta de información que permita mapear capas de aplicación y diagramas [3].

UML está lejos de cubrir las distintas necesidades de proyectos que difieren en su dominio de aplicación, tamaño, necesidades de confiabilidad y presión por tiempos de entrega [10]. Para complementar la generación de código basada en UML, puede ser necesario utilizar DSL y/o pseudocódigo [29]. Los DSL

permiten un énfasis en decisiones de diseño pertinentes al dominio y permiten el empleo de los conceptos y abstracciones más adecuados.

La generación automática de código basada en UML puede disminuir la calidad de los siguientes atributos [29]:

- Capacidad de mantenimiento: a) discontinuidad e incompatibilidad de las nuevas versiones de los GC, b) estructura del código que dificulta su lectura y comprensión.
- Certificación: a) la inspección y análisis de código se dificultan debido a su gran extensión y a la baja comprensibilidad de su estructura. Se han desarrollado GC que mejoraron un diagrama de estado para reducir las líneas de código generadas a partir del mismo [1][28]; b) el código puede depender en tiempo de ejecución, de librerías sin código fuente ni documentación, que puedan introducir vulnerabilidades en los componentes.
- Portabilidad: los estándares MDA del OMG ofrecen portabilidad para nuevo hardware e infraestructuras mediante modelos de software tan complejos que pueden reducir los beneficios del GC.

B. Riesgos o Limitaciones Asociados a la Generación de Código con Herramientas MDA

Se presentan riesgos documentados en la literatura especializada, relacionados con el uso de GC.

El código generado para un clasificador en UML dado, puede ser solo una plantilla de código con estructuras como definiciones de clase, atributos con su tipo de dato, sin código alguno, a llenar por el usuario [3][27]. Puede ser necesaria la inclusión manual de código para la conectividad de base de datos con la interfaz de usuario, cajas de texto, comandos para botones, botones de radio, casillas de chequeo, entre otros, así como para el manejo de tablas de base de datos [3].

Entre los riesgos o limitaciones asociados a herramientas específicas se encuentran:

- Uso de formatos propietarios no estándar para almacenar los diagramas.
- En 2005 existían herramientas con dificultades si un diagrama de estado estaba sobrecargado, lo cual se simplificaba con pseudocódigo logrando así generar menos líneas de código [36]. Esta dificultad puede evitarse convirtiendo el diagrama a un formato XMI u OCL como uno de los pasos iniciales del GC para producir el código.
- Diagramas no procesados [22].

Los riesgos descritos se han tomado en cuenta para proponer el enfoque que se presenta a continuación.

VI. Enfoque Propuesto para Seleccionar GC

Solo algunas herramientas basadas en UML se implementan con una metodología particular [9]. Las actividades presentadas en la Tabla II se proponen como una aproximación hacia un método para la selección de GC, dentro del marco de la MDA, de las fases del Proceso Unificado (UP) y de las prácticas ágiles aplicables en la construcción mediante GC [37].

Estas actividades tienen por objetivo suplementar la insuficiencia de MDA como enfoque para desarrollar software, según se describió en la Sección II. Se proponen tanto para modelos PIM estables como para modelos que evolucionan con varias iteraciones durante la construcción de componentes.

Tabla II: Actividades Propuestas para Seleccionar GC

Modelos disponibles	**Fases del Proceso Unificado**			
	Iniciación	**Elaboración**	**Cons-trucción**	**Tran-sición**
CIM/PIM	**1.** Definir la estrategia para la generación automática de código. **2.** Seleccionar los GC.			
PIM		**3.** Evaluar los GC seleccionados. **4.** Definir el enfoque de integración de los GC.		
PSM		**5.** Generar el código de componentes y pruebas unitarias.		

El gerente del proyecto, los analistas, diseñadores y probadores pueden participar en el enfoque propuesto, pero los responsables principales son una pareja conformada por un modelador y un programador que inician la actividad 1, imitando la organización de equipos ágiles. Pueden conformarse parejas adicionales al avanzar el proceso, por cada paquete o capa a desarrollar.

A. Actividad 1. Definir la Estrategia para la Generación Automática de Código

Esta actividad se presenta en la Figura 1. Su objetivo es asegurar que el proceso para generar código pueda integrarse con las metodologías y herramientas de desarrollo.

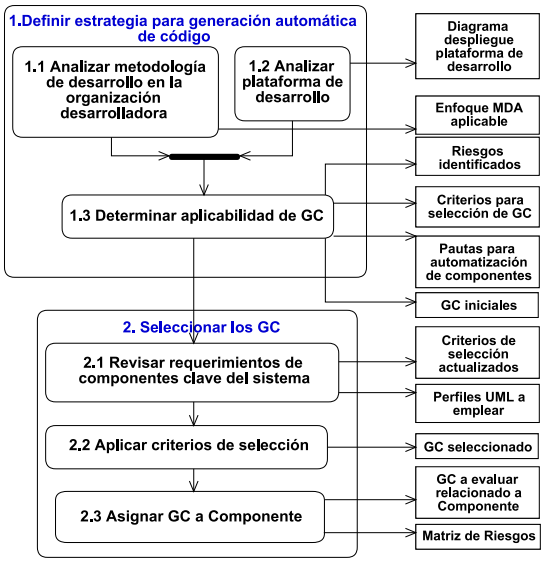

Figura 1: Enfoque Propuesto – Actividades 1 y 2

1) Sub-actividad 1.1. Analizar la Metodología de Desarrollo de la Organización Desarrolladora: Analizar las metodologias y sus métodos, procedimientos, técnicas y herramientas en uso, en relación con el proceso de generación automática de código.

2) Sub-actividad 1.2 Analizar la Plataforma de Desarrollo: Analizar la documentación del software instalado en la plataforma de desarrollo. Se crea un diagrama de despliegue destacando los componentes que forman parte de los procesos de modelado y construcción. Al igual que la sub-actividad anterior, va a facilitar la integración de los enfoques, técnicas y herramientas de desarrollo.

3) Sub-actividad 1.3 Determinar la Aplicabilidad de la Generación Automática de Código Basada en Modelos UML: Determinar cómo se integra la generación automática de código al ambiente de desarrollo, identificando riesgos potenciales, criterios para selección de GCs compatibles, pautas de automatización y una lista inicial con los GC a seleccionar.

Se ejecuta una vez analizados los métodos y herramientas de todas las fases del desarrollo vigentes en la organización, incluyendo las estrategias de adquisición [29], considerando que MDA es apropiado para proyectos grandes ejecutados de una forma muy estructurada y que seleccionar herramientas eficientes y adaptar los procesos de desarrollo existentes a un nuevo proceso de desarrollo ágil y centrado en modelos, puede resultar una tarea muy retadora [10].

Esta sub-actividad incluye determinar si se cuenta con un PIM para construir un prototipo de viabilidad, de bajo costo y desechable, para cada GC a evaluar.

Los productos principales creados en esta actividad son:

- Diagrama similar al de la Figura 2, que represente la integración existente entre las herramientas de modelado, IDE, DBMS y los GC. Varios modeladores UML incluyen un GC [9][23][27].

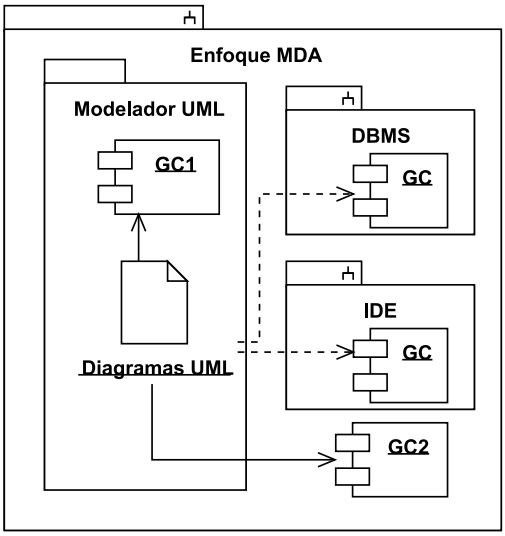

Figura 2: Componentes del Ambiente de Desarrollo

- Enfoque de MDA aplicable: UML ejecutable o transformación. Los modelos UML-ejecutables se consideran el enfoque Agile de MDA que puede reducir la fase de modelado. Deben ser integrados, a diferencia de transformados, para construir la funcionalidad del sistema [13]. La aplicación de UML ejecutable con prácticas Ágiles, debería aumentar la eficiencia del desarrollo del proyecto [10][21].
- Lista de riesgos a mitigar. Posteriormente se refinan y actualizan al crear una matriz de riesgos. El seguimiento de los riesgos relacionados con los componentes en cada sprint o iteración será apoyada por la gestión de riesgos del proyecto global.
- Entre los criterios de selección, se determina si se eligen solo los GC o además las herramientas MDA, para transformaciones Modelo a Modelo.
- Pautas para automatización. Estas incluyen: a) el tipo de componente a generar mediante los GC, indicando su paquete o capa; b) métodos para codificar: mediante GC conformes con estándares MDA, con GCs sin certificaciones de OMG MDA, con DSL, pseudocódigo o en forma manual c) no utilizar GC cuando exista el método probado para la operación de una clase, debido al escaso beneficio de generar el diagrama de actividad para un algoritmo de ordenamiento y codificarlo mediante un GC.

B. Actividad 2. Seleccionar los GC

Esta actividad se presenta en la Figura 1. Su objetivo es seleccionar los GC candidatos a evaluar en la actividad 3. La selección de los GC impacta todo el ciclo de vida de desarrollo de sistemas y, siendo numerosas las herramientas MDA que pueden incluir la funcionalidad de los GC [14][27], se determinan las candidatas a una posterior evaluación.

1) Sub-actividad 2.1 Revisar Requerimientos de Componentes Clave del Sistema a Construir: Mediante el análisis de los requerimientos de los componentes clave del sistema, modelados en los diagramas UML, se determina si los modelos están lo "suficientemente completos" para generar código que permita evaluar los GC.

Para cada componente clave se identifica su capa de aplicación y si su código es de comportamiento, estructura o interacción. Esta información debe estar disponible para la selección de los GC y permitirá actualizar y ampliar los criterios formulados anteriormente, como resultado de analizar los modelos existentes.

Si un componente clave no tiene disponible un PIM, se puede modelar el dominio del sistema con un diagrama de clases y adicionalmente modelar el comportamiento externo del componente con un diagrama de secuencia [12].

2) Sub-actividad 2.2. Aplicar los Criterios de Selección: Asignar pesos a los criterios de selección y aplicarlos a los GC.

Para aplicar los criterios es recomendable disponer del GC instalado en una plataforma que permita generar el código de

componentes clave con base en los diagramas y determinar el nivel de cumplimiento de cada criterio de selección por el GC.

La Tabla III muestra los criterios de selección de GC, entre otros, a aplicar en esta sub-actividad. Estos criterios se proponen de acuerdo a las características descritas en la Sección IV.

Tabla III: Características para la Selección y Evaluación de Generadores Automáticos de Código basados en UML (GC)

Características para seleccionar y evaluar los GC	S	E
Funcionalidad		
- Código generado a nivel de producción: C, C++, Java, C#, Visual Basic, Perl, Python, PHP.	√	
- Código generado para pruebas.		
- Diagramas UML procesados y no procesados o con fallas.	√	√
- Versión de UML: 2.4, 2.1, 2, 1.3.	√	
- "Round-trip" para crear y ajustar diagramas a partir del código.		
- Enfoque de generación de código: estructural, comportamiento, intercambio.	√	√
- Generar código para definir y manipular datos en DBMS específicos.		
- Integración con verificadores de modelos como SPIN.		
- Capacidades de lenguajes de dominio específico (DSL, siglas en inglés).		
- Transformación CIM, PIM y PSM.	√	√
- Ejecución de modelos.	√	√
Compatibilidad con los ambientes de desarrollo y operaciones		
- Plataforma Operativa: Linux, Mac OS X, MS Windows, Android.	√	
- DBMS.		
- Formato de archivo para intercambio de información: OCL, XMI, .uml (de StarUML).		
- Integración con ambientes/herramientas: CVS, Eclipse, Visual Studio.NET., entre otros.		
- Integración con herramientas para transformar modelos CIM-PIM-PSM		√
Otras características		
- Proveedor: código abierto o comercial.	√	
- Uso especializado o de propósito general.	√	
- Continuidad de la herramienta y estabilidad de la versión.	√	

Se recomienda tener en cuenta lo siguiente al aplicar los criterios de selección:

- Aplicar en primer lugar, los criterios a los GC del modelador UML y del IDE,
- En segundo lugar se aplican a los GC de proveedores reconocidos identificados en la actividad 1 y actualizados en la sub-actividad 2.1.
- Los GC que apoyan el desarrollo de sistemas de tres capas - presentación, lógica empresarial y persistencia - pueden ser adecuados para los requerimientos exigentes de sistemas de tiempo real y embebidos, en cuanto a capacidad de mantenimiento, eficiencia, seguridad y tolerancia a fallas, entre otros requerimientos no funcionales.
- La compatibilidad con: a) las herramientas que almacenan los modelos de análisis y con otros métodos y herramientas adoptados por la organización o equipo de desarrollo [29], b) la plataforma de desarrollo y de operaciones incluyendo el IDE y el DBMS y c) estándares OMG MDA incluyendo especificaciones de UML [27].

Para esta selección preliminar, se utilizan los criterios tildados en la columna titulada "S", los cuales tendrían que adaptarse y tener pesos asignados según las actividades anteriores. Esta sub-actividad puede requerir entrevistas a proveedores, aplicación de cuestionarios, análisis de experiencias de otros proyectos, revisión de especificaciones en manuales y el uso del GC.

Los criterios tildados bajo la columna "E" probablemente requieran más recursos y tiempo para evaluar, incluso pueden necesitar modelos PIM estables para componentes críticos.

La Figura 3 representa los criterios considerados más determinantes para seleccionar los GC, debido a la estrecha relación que estos tienen con el código generado. Se consideran criterios relacionados con la idoneidad funcional de acuerdo con los conceptos, prácticas y funcionalidades de GC, descritos en las secciones anteriores. Se recomienda aplicar los criterios de selección de los GC en cuanto a los tres tipos de código y capas identificadas para los componentes en la sub-actividad 2.1, considerando el nivel del modelo disponible.

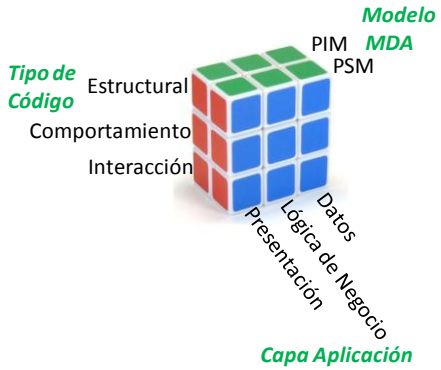

Figura 3: Criterios de Idoneidad Funcional para Seleccionar Generadores de Código

Si existen componentes a construir con código de comportamiento, debido a que este código se genera a partir de los diagramas de máquinas de estado, de componentes, de secuencia y de actividad [27], los GC se seleccionan si procesan estos diagramas.

2) Sub-actividad 2.3. Asignar GC a Componentes: Esta asignación se apoya en las especificaciones del GC.

UJECTOR es un GC que especifica la siguiente relación entre diagramas UML 2.0 y código fuente [28]: de clase, para generar el esqueleto del código Java; de secuencia, para los métodos y finalmente, de actividad y de secuencia, para la manipulación de objetos e interacción con el usuario.

En esta actividad se determina si el GC será asignado a uno o más componentes para ser evaluado. Se identifican riesgos y limitaciones como es el caso de diagramas no procesados por el GC. Puede ejecutarse durante la creación del PIM. Requiere identificar todos los diagramas que representan cada capa en el PIM y haber determinado si el GC los procesa utilizando para ello, las capacidades del GC para transformar automáticamente el PIM en uno o más PSM.

Los productos generales resultantes de la actividad 2 son:

- Los criterios para seleccionar los GC basados en sus características y capacidades.
- Perfiles de UML a emplear como es el caso de: perfil ITU-T Z.109 de UML para sistemas distribuidos; MARTE para sistemas embebidos de tiempo real; jUML y fUML para xUML ejecutable [2].
- La asociación Componente-GC para los componentes clave. Esta se representa en una matriz la cual se refinará en las actividades siguientes, revisando de nuevo los aspectos anteriores al contar con el modelo PIM completo.
- Matriz de riesgos basada en la lista preliminar obtenida en la actividad 1, actualizada de acuerdo a los mencionados en la Sección V para los GC.

C. *Actividad 3. Evaluar los GC Seleccionados*

Puede ejecutarse en forma paralela a la anterior; su objetivo es determinar si los GC seleccionados generan código correcto y completo para cada capa y tipo de código (comportamiento, estructura e interacción) del sistema a construir. Se indican en la Figura 4 sus sub-actividades y productos:

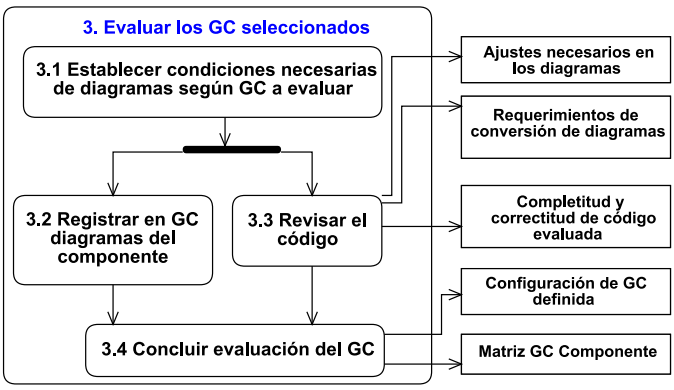

Figura 4: Enfoque Propuesto – Actividad 3

1) Sub-actividad 3.1. Establecer las Condiciones Necesarias en los Diagramas según los GC a Evaluar: Esta subactividad requiere que el PIM esté disponible. Los diagramas UML recomendados a continuación, podrán ser utilidad para analizar el PIM y luego completarlo:

- Los relacionados a los tipos de código de acuerdo con el OMG UML [9],
- Los relacionados a las 4+1 vistas de la organización de un sistema de software [19] y
- Los indicados en la Tabla I [18].

Luego se determinan las condiciones que deben reunir los diagramas para un GC particular. El UP recomienda evitar detalles innecesarios con demasiada anticipación a la construcción [11]. Esto puede resultar en un modelo con diagramas que habrán de ser ajustados a las condiciones que imponen los GC.

Puede ser necesario completar los modelos con el apoyo de la ingeniería en reverso, transformando el código que cumpla los requerimientos del sistema a desarrollar. El modelo resultante puede compararse con el PIM incompleto que se obtuvo en la ingeniería hacia adelante, a fin de completar sus diagramas.

2) Sub-actividad 3.2. Registrar en el GC los Diagramas del Modelo que Representa el Componente a Codificar: Unos GC generan código para paquetes de diagramas además de generar código para un diagrama específico.

La documentación de los GC puede describir la relación entre los componentes frecuentemente usados, como son los componentes de las capas de aplicaciones Web, y los diagramas indispensables como parte de un modelo.

3) Sub-actividad 3.3. Revisar el Código Generado: Se ejecuta mediante inspecciones que determinan el porcentaje generado y si cubre los requerimientos representados en el modelo.

Las inspecciones incluyen: a) el nivel de adecuación para los tipos de código y para estándares de calidad según los requerimientos; b) la consistencia entre el diagrama y el código y c) el número y complejidad de los ajustes requeridos en los diagramas.

Las sub-actividades 3.2 y 3.3 son iteraciones modelado-codificación-prueba para evaluar tanto el código generado como la facilidad de ejecución de las iteraciones.

4) Sub-actividad 3.4. Concluir la Evaluación: Decidir e informar si el GC va a generar el tipo de componente para el cual fue evaluado.

Se requiere la siguiente información actualizada: a) el componente a desarrollar, la capa del sistema a la cual pertenece, el tipo de enfoque de su código y todos los diagramas UML que lo representan y b) los ajustes a realizar en los diagramas UML según los requerimientos que impone el GC.

Los productos principales generados en esta actividad son:

- los ajustes en los diagramas indispensables para generar código, considerando cómo el GC implementa los conceptos de UML
- requerimientos de conversión de los diagramas para su procesamiento por un GC
- el nivel de completitud y correctitud del código generado; será necesario contar con el código de las pruebas para determinar si el código de los componentes clave funciona correctamente
- la configuración de cada GC con el cual se van a crear los componentes y sus pruebas
- matriz GC-Componente actualizada indicando el porcentaje de código, diagramas sin soporte del GC, aspectos por resolver y los requerimientos de conversión.

D. *Actividad 4. Definir la Integración de los GC*

Las sub-actividades y productos de esta actividad se presentan en la Figura 5. El objetivo es definir cómo se integran los métodos y herramientas de forma que esté disponible el ambiente de desarrollo necesario para generar el código.

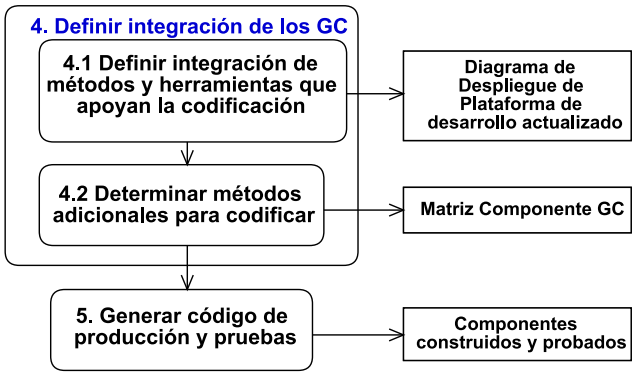

Figura 5: Enfoque Propuesto – Actividades 4 y 5

1) Sub-actividad 4.1. Definir la Integración de Métodos y Herramientas que Apoyan la Codificación: Definir cómo se integran los GC entre sí y con las demás herramientas de las plataformas de desarrollo y de producción.

Esta sub-actividad atiende a la necesidad de utilizar varias herramientas para generar el código completo de diversos componentes. Puede completarse durante la ejecución de las actividades 1 y 3, para sistemas sencillos de baja complejidad y componentes poco diversos.

2) Sub-actividad 4.2. Determinar el Mapeo Componente-Otro Método de Codificación: Actualizar la matriz Componente-GC, tomando en cuenta la metodología del proveedor y los procesos de conversión necesarios.

En la actividad anterior, al concluir la evaluación del GC, se identificaron los componentes sin apoyo de GC, los cuales se crean mediante métodos a documentar en esta sub-actividad.

Los productos principales generados en esta actividad son:

- Diagrama similar al de la Figura 2 que represente la nueva integración entre herramientas de modelado, IDE, DBMS y GC.
- Matriz Componente-GC actualizada con el método de codificación definitivo: GC, DSL, pseudocódigo, programación manual, entre otros. La opción para generar código se registra en la matriz creada en la actividad 2, y actualizada en la actividad 3.
- Opcionalmente pueden generarse la actualización de la configuración de los GC y la creación de nuevos procedimientos de integración.

E. Actividad 5. Generar el Código Fuente de los Componentes y de las Pruebas

Actividad opcional presentada en la Figura 5; su objetivo es generar el código de los componentes y de las pruebas mediante las herramientas identificadas en la actividad 3 e integradas en la actividad anterior.

Aunque la evaluación del GC para un componente clave específico se completó en la actividad 3, cuando se trata de componentes críticos para la misión de la organización o si la configuración del GC pudiera variar después de la actividad 4, puede ser necesario generar la totalidad del código del componente como parte de su evaluación definitiva.

El producto que resulta de esta actividad es el código generado para los componentes clave, incluyendo el código de pruebas, de forma que pueda afirmarse que el componente se codificó y se probó. Al generar el código se deben tomar en cuenta las siguientes recomendaciones:

- Se genera el código mediante el uso del GC sin necesidad de completar un gran diseño inicial. La generación de código se puede iniciar con modelos en un estado "suficientemente bueno" los cuales serán completados durante el uso de los GC. Esto tiene como ventaja adicional, que permite la detección oportuna de fallas en requerimientos suplementarios o no funcionales como el desempeño [34].
- Solo en el proceso de generación se agregan elementos dependientes de la plataforma. Cuando la tecnología cambia solo se requiere actualizar el GC, pero los modelos que definen la aplicación pueden ser directamente reutilizados [10].
- Los diagramas se ajustan hasta obtener código completo y correcto. Para asegurar la calidad de la programación, se modifica el diagrama antes que el código.
- Las funciones de ingeniería en reverso y "round-trip" permiten verificar la consistencia entre el código fuente generado y los diagramas, y también, mantener los diagramas actualizados en forma automática al actualizar el código.
- Modelando en pareja por un sprint y con rotación de parejas para detectar y resolver problemas de modelado en forma instantánea, se puede ejecutar esta actividad aplicando las prácticas recopiladas que combinan metodologías ágiles y MDA [17], descritas en la Sección III.

Una vez presentadas las cinco actividades propuestas, cabe agregar que al igual que XP y ASD evitan distinguir entre las actividades de diseño y las de implementación [10], las actividades anteriores se ejecutan de acuerdo con el avance del desarrollo de los modelos, y no con referencia a flujos de trabajo propios del Proceso Unificado.

Las actividades descritas se pueden adaptar con enfoques de prototipos de diseño que evolucionen hacia prototipos de producción o implementación.

El enfoque propuesto puede evolucionar mediante su prueba en un proyecto piloto. Se revisaron aspectos mejorables identificados al contrastar el enfoque con buenas prácticas recomendadas para el desarrollo de metodologías [38]. De esa evaluación se identificaron mejoras referidas a la completitud de los elementos involucrados y a la documentación. En trabajos futuros se implementarán estas mejoras, entre otras. Se considera que el enfoque puede guiar selecciones de GC en proyectos de baja complejidad para el desarrollo de sistemas sin criticidad alta para la misión de la organización del usuario.

VII. CONCLUSIONES

Hoy en día, las herramientas MDA disponibles presentan niveles distintos de conformidad con estándares del OMG; adicionalmente, varían en cuanto a las funciones que ofrecen, lo cual hace necesaria una selección rigurosa y exhaustiva que pueda seguir un proceso factible de instanciar.

En la actualidad, las herramientas MDA están más cerca de proveer altos niveles de automatización y reusabilidad aunque sin suministrar la totalidad del código, particularmente el código de comportamiento. Se requieren varios modelos en distintos lenguajes de modelado, e incluso código en lenguajes de programación para desarrollar un sistema de software completo [2].

La generación automática de código basada en modelos cambia el diseño tanto o más que la implementación, y coloca el énfasis más en los distintos niveles de modelos que en la programación. Aunque los GC pueden seleccionarse principalmente por sus capacidades para generar código con base en diagramas UML, un criterio que recientemente tiende a ser más aceptado es la capacidad de transformaciones modelo a modelo y modelo a código.

Los riesgos señalados podrán superarse a medida que evolucione la MDA y sus herramientas. Existen desarrollos de sistemas críticos para la misión que consideran la generación automatizada de código basada en UML como más confiable y segura, sobre todo si cumple con los estándares de calidad del OMG MDA. Al evolucionar y mejorar los GC, la codificación manual pudiera considerarse riesgosa e injustificada y la verificación para certificar una aplicación podrá requerir el uso de los GC compatibles con dichos estándares.

Por el contrario, pudiera repetirse lo ocurrido con los CASE hace más de una década, en cuanto a la falta de apoyo en las transformaciones necesarias para derivar PSM a partir de modelos PIM. Varias herramientas realizan transformaciones directamente de un modelo PIM a código fuente mediante un lenguaje de plantilla en forma inapropiada. Sería de esperar que la iniciativa MOF 2.0/QVT del OMG progrese hacia la estandarización de las transformaciones de modelos [31].

Desde hace décadas, los usuarios disponen de sistemas con componentes configurables en su totalidad y de DBMS que proporcionaban funcionalidad completa para sistemas de procesamiento de transacciones. Dentro de entornos donde se dispone de esas soluciones, los proveedores de GC se adaptan para responder a requerimientos fuertes y propios de sistemas de tiempo real, embebidos, complejos y de alta criticidad. Esa adaptación se apoya en la MDA, la cual continua su evolución hacia lo que se espera sean niveles de madurez y de aceptación por la comunidad de desarrolladores.

Las prácticas ágiles pueden reducir el potencial de los riesgos causados por procesos pesados de desarrollo que se consideran parte del enfoque MDA [17]. Esto debe tomarse en cuenta dentro del proceso de gestión de riesgos del proyecto.

Como trabajo futuro se prevé comparar el código producido por distintas herramientas GC y ajustar el enfoque diseñado, según los resultados de esa comparación y según las prácticas sugeridas por los proveedores de esas herramientas.

REFERENCIAS

[1] M. Rincón, J. Aguilar, F. Hidrobo, *Generación Automática de Código a Partir de Máquinas de Estado Finito*, Universidad de los Andes, Mérida, Venezuela. Computación y Sistemas, vol. 14, no. 4, pp. 405-421, ISSN 1405-5546, 2011.

[2] M. Karanam, *MDA Tool Support for Model Driven Software Evolution: A Survey*, International Journal of Computer Science Engineering (IJCSE), ISSN: 2319-7323, vol. 4, no. 01, January 2015.

[3] S. D. Rathod, *Automatic Code Generation with Business Logic by Capturing Attributes from User Interface via XML*, International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Channai, India, pp. 1480-1484, 2016.

[4] M. Asadi, M. Ravakhah, R. Ramsin, *An MDA-based System Development Lifecycle*, Second Asia International Conference on Modelling & Simulation, Asia International Conference on Modelling & Simulation, Kuala Lumpur, Malaysia, DOI:10.1109/AMS.2008.19, 2008.

[5] C. M. Zapata, J. J. Chaverra, *Una Mirada Conceptual a la Generación Automática de Código*, Revista EIA, Escuela de Ingeniería de Antioquia, Medellín, Colombia. ISSN 1794-1237, no. 13, pp. 143-154, Julio 2010.

[6] A. Noureen, A. Amjad, F. Azam, *Model Driven Architecture - Issues, Challenges and Future Directions*, Journal of Software, vol. 11, no. 9, September 2016.

[7] OMG, Object Management Group, *Model Driven Architecture (MDA), MDA Guide rev. 2.0*, OMG Document ormsc/2014-06-01, 2014.

[8] S. Motogna, I. Lazar, B. Parv, I. Czibula, *An Agile MDA Approach for Service-Oriented Components*, Electronic Notes in Theoretical Computer Science 253, 95–110, 2009.

[9] UML.org. http://www.uml.org/what-is-uml.htm.

[10] B. Rumpe, *Agile Modeling with the UML*, Radical Innovations of Software and Systems Engineering in the Future. 9th International Workshop, RISSEF 2002. Venice, Italy, October 2002.

[11] S.R. Schach, *Análisis y Diseño OO con UML y el Proceso Unificado*, 4ta edición, McGraw Hill, 2005.

[12] B. Bousetta, O. El Beggar, T. Gadi, *A Methodology for CIM Modelling and its Transformation to PIM*, Journal of Information Engineering and Applications. vol. 3, no. 2, 2013.

[13] A. El-Sheikh, A. Omran, *Suggested Framework for Agile MDA and Agile Methodologies*, The Research Bulletin of Jordan ACM, ISSN: 2078–7952, Vol. II (III) 2011.

[14] T. Calic, S. Dascalu, D. Egbert, *Tools for MDA Software Development: Evaluation Criteria and Set of Desirable Features*, IEEE DOI 10.1109/ITNG.2008.241, © 2008.

[15] R. Pilitowski and A. Dereziñska, *Code Generation and Execution Framework for UML 2.0 Classes and State Machines.* In: Sobh T. (eds) Innovations and Advanced Techniques in Computer and Information Sciences and Engineering. Springer, Dordrecht, 2007.

[16] M. Rahmouni and S. Mbarki, *Model-Driven Generation: From Models to MVC2 Web Applications*, International Journal of Software Engineering and its Applications, vol. 8, no. 7, pp. 73-94, 2014.

[17] S. Hansson and Y. Zhao. *How MAD Are We? Empirical Evidence for Model-Driven Agile Development*, Bachelor of Science Thesis, Software Engineering and Management. University of Gothenburg, Chalmers University of Technology, Department of Computer Science and Engineering, Göteborg, Sweden, June 2014.

[18] S. W. Ambler, http://www.agiledata.org/essays/enterpriseArchitecture Techniques.html. Ambysoft Inc. Copyright 2002-2013.

[19] FCG Software Services (FCGSS), *Applying 4+1 View Architecture with UML 2*, White Paper, Copyright ©2007.

[20] J. B. Quintero, R. Anaya, *MDA y el Papel de los Modelos en el Proceso de Desarrollo de Software*, Revista de la Escuela de Ingeniería de Antioquía. no. 8, Julio/Diciembre, 2007.

[21] B. Rumpe, *Executable Modeling with UML - A Vision or a Nightmare?* - Issues & Trends of Information Technology Management in Contemporary Associations, Seattle. Idea Group Publishing, Hershey, London, pp. 697-701, 2002.

[22] A. Mateen, A. Tabassum, *A Framework for Model Driven Transformation Engineering towards Software Architecture and Performance*, International Journal of Computer Applications, vol. 143, no. 8, June 2016.

[23] A. Ramirez, P. Vanpeperstraete, A. Rueckert, K. Odutola, J. Bennett, L.Tolke, and M. van der Wulp, *ArgoUML User Manual: A Tutorial and Reference Description. Version 0.34 of ArgoUML*. http://argouml-downloads.tigris.org/nonav/argouml-0.34/manual-0.34.pdf. Copyright © 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011.

[24] StarUML. *StarUML 5.0 Developer Guide* http://staruml.sourceforge.net/docs/StarUML_5.0_Developer_Guide.pdf.

[25] Altova Mapforce, http://www.altova.com/umodel/editions.

[26] Acceleo. *Guía del Usuario,Versión 3.1.0*. http://www.obeonetwork.com/group/acceleo/page/acceleo-3-1-0-user-guide.

[27] H. Eichelberger, Y. Eldogan, K. Schmid, *A Comprehensive Analysis of UML Tools, their Capabilities and their Compliance*, Software Systems Engineering. Universität Hildesheim. August 2011.

[28] M. Usman, A. Nadeem, *Automatic Generation of Java Code from UML Diagrams using UJECTOR*, Center for Software Dependability, Mohammad Ali Jinnah University, Islamabad, Pakistan. International. Journal of Software Engineering and Its Applications, vol. 3, no. 2, April 2009.

[29] J. Klein, H. Levinson, J. Marchetti, *Model-Driven Engineering: Automatic Code Generation and Beyond*, Technical Report. CMU/SEI-2015-TN-005, Software Solutions Division. http://www.sei.cmu.edu, March 2015.

[30] J. de Lara, E. Guerra, J. Sánchez, *Model-Driven Engineering with Domain-Specific Meta-Modelling Languages*, Software & Systems Modeling, vol. 14, no. 1, pp. 429-459, February 2015.

[31] J. Bettin, *Best Practices for Component-Based Development and Model-Driven Architecture*, Version 1.0, White Paper, SoftMetaWare Ltd. August 2003.

[32] A. Jakimi and M. El Koutbi. *An Object-Oriented Approach to UML Scenarios Engineering and Code Generation*, International Journal of Computer Theory and Engineering, vol. 1, no. 1, April 2009.

[33] D. Haywood, http://www.theserverside.com/news/1365166/MDA-Nice-idea-shame-about-the. The server Side. Your Enterprise Java Community, 2004.

[34] P. Guha, K. Shah, S. Shankar, P. Shukla, S. Singh, *Incorporating Agile with MDA Case Study: Online Polling System*, International Journal of Software Engineering & Applications (IJSEA), vol. 2, no. 4, DOI: 10.5121/ijsea.2011.2408 83, October 2011.

[35] S. L. Jim, *From UML Diagrams to Behavioural Source Code*, Thesis Universiteit van Amsterdam, Centrum voor Wiskunde en Informatica, 2006.

[36] I. Lipkin and A. K. Huber, *UML Design and Auto-Generated Code: Issues and Practical Solutions*, The Journal of Defense Software Engineering, November 2005.

[37] M. Picón, C. Maldonado, *Generación Automática de Código Basada en Modelos UML*, Cuarta Conferencia Nacional de Computación, Informática y Sistemas (CoNCISa 2016), pp. 134–138, Caracas, Venezuela, 2016.

[38] J. Whitten, L. Bentley, *Systems Analysis and Design Methods*. 7th edition, McGraw-Hill, 2007.