YAAMDDA: Una propuesta AMDD para el desarrollo de Aplicaciones Web Enterprise Enriquecidas de Internet

Víctor Cajes, Magalí González, and Luca Cernuzzi

Facultad de Ciencias y Tecnología,
Universidad Católica "Nuestra Señora de la Asunción", Asunción, Paraguay
{vcajes, mgonzalez, lcernuzz}@uca.edu.py
http://www.dei.uc.edu.py/

Resumen Las Aplicaciones Web Enterprise (WEB-EAS), y en particular aquellas basadas en tecnología RIA (Rich Internet Application), son de gran importancia hoy en día, ya que ayudan a administrar y controlar completamente una organización, excediendo los límites físicos de la misma. Este tipo de aplicaciones pueden ser construidas siguiendo diversos enfoques del Desarrollo Dirigido por Modelos Ágil (AMDD) con el fin de agilizar el proceso de desarrollo mediante la utilización de técnicas de modelado, gestión ágil de proyectos y generación automática de código.

La contribución principal de este estudio consiste en YAAMDDA, un innovador enfoque y soporte computacional basado en el enfoque AMDD, en donde a partir de un simple modelo UML es posible generar de forma automática una aplicación funcional, agilizando así el proceso de desarrollo de las RIAs WEB-EAS. Se presenta además un caso de estudio que ofrece una primera validación de la propuesta YAAMDDA.

1. Introducción

De la diversa variedad de sistemas software actuales, las Aplicaciones Enterprise (EAS) son muy importantes, pues ayudan a administrar por completo a una organización [1]. Entre ellas, una gran diversidad están siendo desarrolladas para ejecutarse en un ambiente Web (WEB-EAS). En lo que refiere a ambientes Web, por su parte, las *Aplicaciones Enriquecidas de Internet (RIAs)* se están posicionando muy velozmente. Esto se debe al hecho de que permiten la simulación de aplicaciones de escritorio en un ambiente Web, permitiendo que los datos puedan ser procesados tanto en el cliente como en el servidor para que el cliente pueda responder correctamente a las operaciones de cálculo, actualizaciones de interfaz y las interacciones con el usario [2]. Esto permite una mejor experiencia a los usuarios, a la hora de utilizar un sistema dado [3]. Estas tendencias y tecnologías actuales llevan consigo la necesidad de contar con procesos de desarrollo adecuados para obtener aplicaciones de calidad.

Uno de los principales desafíos de los desarrolladores de software consiste en alcanzar el nivel necesario de agilidad para adecuarse al mercado. Esto hace que sea necesario reducir los costos, adaptarse rápidamente a los cambios que surjan y, por sobre todo, hacer entregas a tiempo de los productos solicitados. Los métodos Ágiles surgen como una alternativa muy interesante, ya que presentan guías y procedimientos para evitar múltiples problemas detectados con los métodos tradicionales [4], guiando así a los proyectos hacia el éxito buscado [5], [6].

Otras experiencias han mostrado también que el *Desarrollo Dirigido por Modelos (MDD)*, permite acelerar sustancialmente los desarrollos de los sistemas. En este sentido, cabe resaltar que en algunos casos reales de éxito en ambiente enterprise, se han logrado aumentos significativos en la productividad [7], [8]. Esta mejora se da como consecuencia de que se evita el desperdicio del trabajo humano en ciertas tareas repetitivas, además de permitir que un computador realice de forma automática e instantánea ciertas tareas que anteriormente eran realizadas de forma manual [9].

En este sentido, suena sumamente interesante poder unificar los conceptos de métodos ágiles y MDD con el fin de promover el ágil desarrollo de aplicaciones WEB-EAS, manteniendo la calidad de los productos, la flexibilidad de la plataforma y que estos puedan desarrollarse en el menor tiempo y al menor costo posible. El *Desarrollo Dirigido por Modelos Ágil (AMDD)* [10] es una propuesta que permite unificar las ventajas obtenidas con la adopción de MDD y los métodos ágiles. Ambler menciona que existen tres categorías de enfoques para aplicar AMDD a los proyectos [10]:

1. AMDD Manual (también conocido como modelado ágil): en donde herramientas simples y modelos inclusivos se utilizan para el modelado. El 70-80 % de los equipos desarrolladores lo adoptan.

2. CASE Ágil: en donde herramientas sofisticadas se utilizan para el diseño detallado. La propuesta YAAMDDA se clasifica dentro de esta categoría. Aproximadamente un 20 % de los equipos desarrolladores adoptan ésta modalidad.

3. MDA Ágil: en donde las herramientas de modelado y transformación se basan en el estándar MDA. Aproximadamente un 5 % de los equipos desarrolladores lo adoptan.

Este artículo presenta el enfoque YAAMDDA (Yet Another Agile Model Driven Development Approach), una propuesta AMDD, categorizada como 'CASE Ágil', para la generación de aplicaciones RIAs WEB-EAS. La propuesta, además de incluir aspectos metodológicos, lo complementa con una serie de herramientas que han sido desarrolladas para cubrir todas las necesidades. Se presentan además los resultados de un caso de estudio utilizado para validar la propuesta.

El presente artículo se estructura de la siguiente manera. La sección *2* presenta la propuesta YAAMDDA. La sección *3* discute una primera validación de la

propuesta con un caso de estudio. En la sección *4* se presentan los trabajos relacionados. La sección *5* presenta las conclusiones y trabajos futuros.

2. YAAMDDA: UNA PROPUESTA AMDD PARA UN AMBIENTE WEB ENTERPRISE ENRIQUECIDO

El enfoque *YAAMDDA*, de sus siglas en inglés *Yet Another Agile Model-Driven Development Approach*, fue desarrollado en base a los conceptos que definen al método ágil AUP (REFERENCIA a AUP!!!)y su enfoque AMDD, brindando finalmente un soporte computacional para el desarrollo de WEB-EAS. Para lograr este objetivo fue diseñado modularmente en tres componentes fundamentales bien diferenciados y a la vez muy relacionadas entre sí: *el metamodelo*, *el CASE* (YAACASE) y *el traductor* (YAALATOR).

A continuación se describen en detalle cada uno de los tres componentes que conforman al enfoque *YAAMDDA*, así como el proceso propuesto.

2.1. El proceso YAAMDDA

YAAMDDA está orientado fundamentalmente al desarrollo de RIAs Enterprise, mediante el uso de una herramienta específicamente construida para facilitar y agilizar dicha tarea. La propuesta *YAAMDDA* se puede encuadrar en AMDD como una propuesta Ágil suportada por una herramienta CASE.

Como se puede observar en la figura 1, todo parte de la necesidad de desarrollar una WEB-EAS. AUP establece que en su primera fase del ciclo de vida, denominada *Comienzo*, se deben realizar las primeras reuniones con los clientes con el fin de identificar el alcance del proyecto, se comienza a pensar en una potencial arquitectura para el sistema y se concluye con la aceptación o rechazo de la propuesta de solución por parte de los interesados. Esta fase contempla las siguientes tareas:

1. **Modelado**: comprender el negocio de la organización e identificar una posible solución viable que trate el dominio del problema planteado.
2. **Administración**: se detectan los posibles riesgos que puedan dificultar el proceso de desarrollo, se organizan los recursos humanos y equipos de trabajo, y se define un calendario a largo plazo con las tareas a realizar.
3. **Entorno**: en donde se deben acercar a los equipos de trabajos toda la documentación acerca de las buenas prácticas y estándares a seguir durante el proceso de desarrollo.

Una vez concluida la primera fase, los desarrolladores ya tienen una mejor idea del dominio del problema y, en gran medida, de la lógica de negocios que sigue la organización. Es por eso que la segunda fase del método AUP, denominada

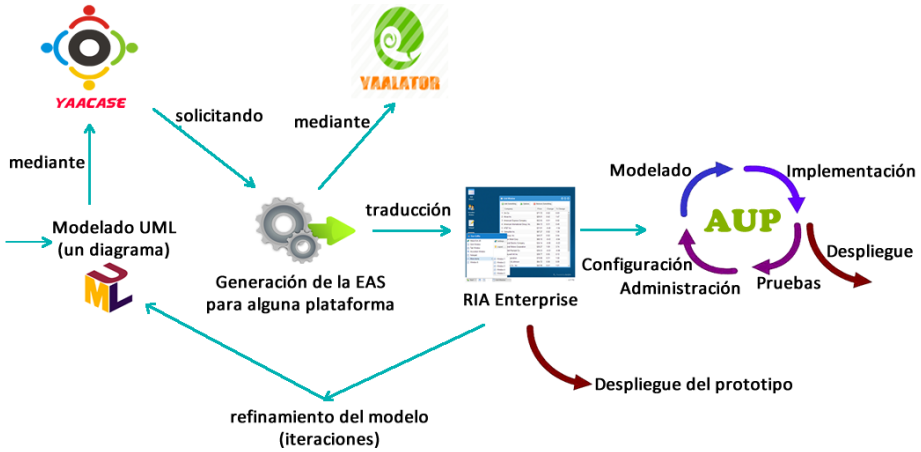

Figura 1: La propuesta YAAMDDA

Elaboración, tiene como objetivo fundamental definir concretamente la arquitectura, y por tanto la plataforma del sistema a utilizar. Durante esta segunda fase, se realiza el modelado del sistema utilizando la herramienta *YAACASE* generando un único diagrama de clase UML enriquecido con estereotipos. Se comienzan a aplicar los estereotipos que se consideren apropiados para el caso, y finalmente se genera automáticamente el primer prototipo de la aplicación. Este prototipo debe ser validado con los clientes.

Teniendo definida la arquitectura, y habiendo desplegado un prototipo funcional y recopilado retroalimentación de la misma por parte de los clientes, comienza la tercera fase del ciclo de vida, denominada *Construcción*, cuyo objetivo fundamental consiste en el desarrollo regular y gradual del software, de tal forma a que responda a las necesidades de mayor prioridad de los interesados en el proyecto. Pero a diferencia de la tercera fase tradicional AUP, en la propuesta *YAAMDDA* se tienen bien diferenciadas dos partes:

1. **Primera Parte**: cómo se observa en el centro de la figura 1, se realizan únicamente modificaciones y refinamientos al modelo construido con la herramienta *YAACASE*, con el fin de ir mostrando a los clientes, en las diversas iteraciones, los avances, cambios y prototipos generados. Una vez aprobado el esquema y prototipo por los clientes, se culmina con la primera parte de la fase de *Construcción*, teniendo como resultado una gran porción del código fuente de toda la aplicación.

2. **Segunda Parte**: cómo se observa en el sector derecho de la figura 1, a partir del código ya generado por *YAALATOR* se comienza a seguir la fase de *Construcción* original del método ágil AUP. Y por tanto se deben realizar las siguientes disciplinas de manera iterativa:

- **Modelado**: se determinan las nuevas historias de usuario a realizar a partir de la retroalimentación obtenida por parte de los clientes, con el fin de satisfacer los requisitos y/o adaptarse a los cambios que surjan.
- **Implementación**: escribir el código fuente necesario para ir satisfaciendo las historias de usuario preparadas previamente.
- **Pruebas**: se realizan las pruebas de unidades implementadas y en su defecto, la refactorización del código cuando haga falta.
- **Despliegue**: pueden haber dos tipos de producciones al terminar una iteración; un despliegue para realizar las pruebas/validaciones correspondientes, o bién el despliegue en producción para usar la aplicación.
- **Configuración**: se lleva un registro del historial de los cambios más relevantes realizados en los artefactos del sistema, con el fin de tener documentado las distintas versiones y variaciones ocurridas durante el proceso de desarrollo.
- **Administración**: se analizan los riesgos (temporales, económicos, tecnológicos, etc.), se definen las eventuales nuevas historias de usuario requeridas y se establece, a partir de la prioridad de las mismas, cuáles serán implementadas en la siguiente iteración.

Las iteraciones de la tercera fase terminan cuando se dispone de una versión de la aplicación estable, funcional y, por sobre todo, que llene las expectativas de los clientes. A partir de ese momento comienza la cuarta y última fase denominada *Transición*, en donde se comienzan a realizar los mantenimientos necesarios. Para el efecto, se realizan tareas similares a aquellas de la tercera fase, pero requiriendo generalmente una cantidad menor de esfuerzo.

En forma general *YAAMDDA* representa la idea propuesta por Ambler en la descripción de su *Proceso Unificado Ágil*, en donde se resalta que *YAAMDDA* es un enfoque "serial en lo largo", a causa de las cuatro fases ordenadas de su ciclo de vida, e "iterativo en lo pequeño" ya que cada una de las fases contiene disciplinas que son ejecutadas de manera iterativa e incremental.

2.2. El metamodelo, los perfiles UML y el soporte computacional

El metamodelo se basa en la arquitectura de metamodelado *MOF (Meta-Object Facility)*. Para el modelado del dominio específico de las RIAs WEB-EAS se propone una extensión del lenguaje UML con el uso de los perfiles UML. En el lado izquierdo de la figura 2 se observa el *el metamodelo* para la definición de la sintaxis abstracta, y en el lado derecho, el perfil UML para la definición de la sintaxis concreta. Como se puede notar, el metamodelo es simple y en general, un proyecto dado consta de un *YAAMDDA Model* (*modelo*), el único diagrama realizado para modelar los requisitos. Este modelo, a su vez puede contener uno o más *Modules* (*módulos*), los cuales a su vez pueden tener uno o más

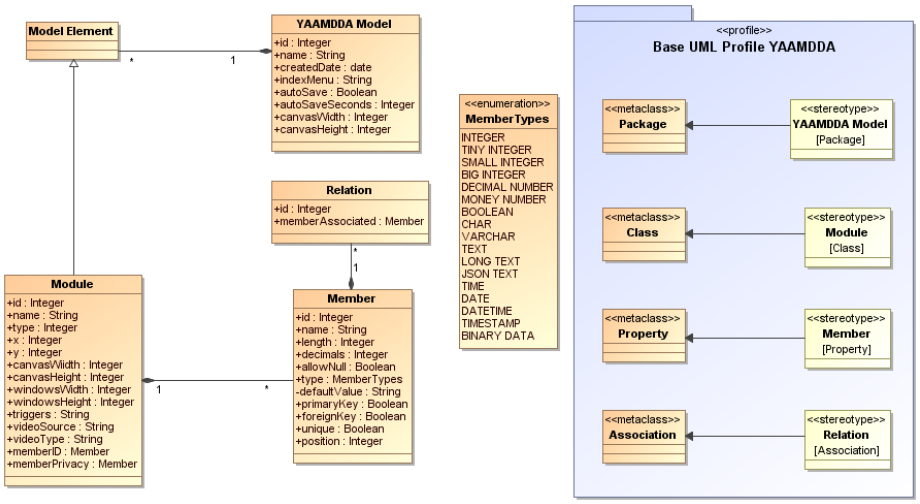

Figura 2: El metamodelo propuesto por YAAMDDA.

Members (*miembros*), los que podrían tener asociados una *Relation* (relación) con cualquier otro miembro del modelo.

Los nombres utilizados en los atributos de las clases del metamodelo fueron elegidos de tal forma a facilitar la comprensión de los mismos. Con el fin de poder modelar cuestiones específicas relacionadas a las WEB-EAS, fueron utilizados los Perfiles UML para extender el diagrama de clases tradicional, y poder asignar *estereotipos* (etiquetas) que den un significado particular a las instancias modeladas. En la figura 3 se observan todos los estereotipos relacionados a la metaclase *Class* y, por lo que ya vimos en la figura 2 del metamodelo, podrán ser asignados a cualquier *módulo* del modelo.

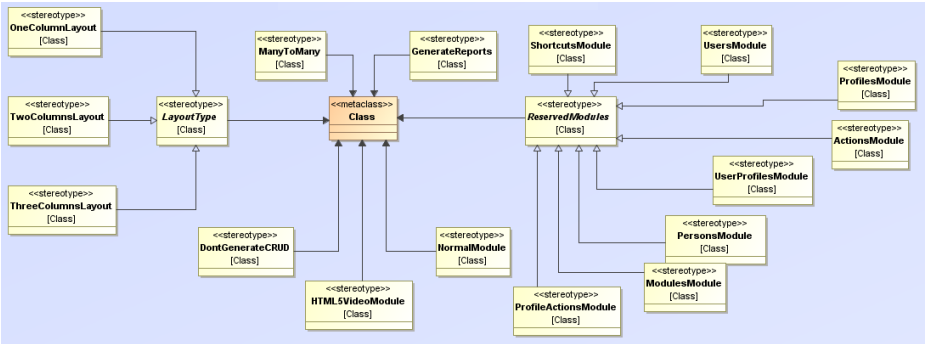

Figura 3: Estereotipos asociados a la metaclase UML *Class*.

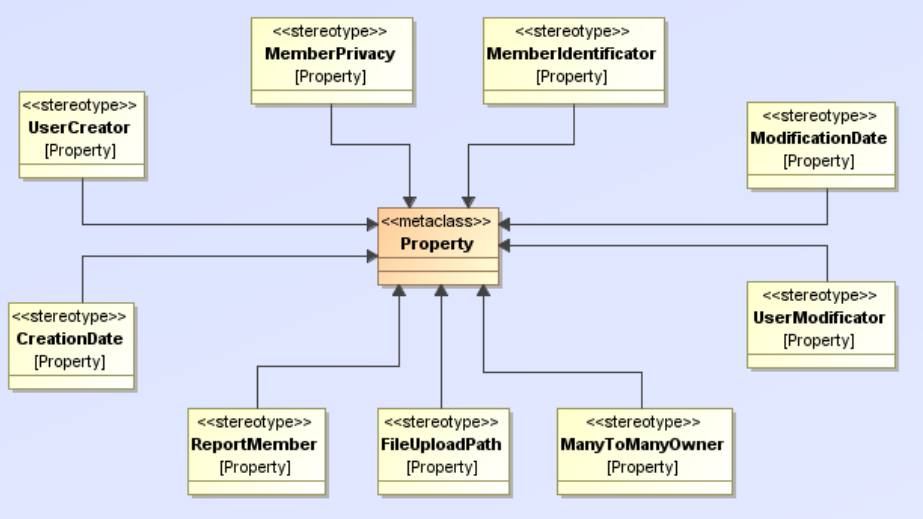

Figura 4: Estereotipos asociados a la metaclase UML *Property*.

En la figura 4 se observan todos los estereotipos relacionados a la metaclase *Property*, que podrán ser asignados a cualquiera de los miembros de cualquier módulo. Finalmente, los dos únicos estereotipos relacionados con la metaclase *Association*, son ComboSelect y SearcherSelect. Ambos se encuentran relacionados exclusivamente a la manera en que los usuarios podrán manipular dichas relaciones.

La herramienta *YAACASE*, de sus siglas de inglés, *Yet Another Agile Computer Aided Software Engineering* facilita a los desarrolladores la tarea de modelado del dominio específico de las WEB-EAS. *YAACASE* está totalmente vinculada con el metamodelo, puesto que facilita la interacción de los diversos componentes (modelos, módulos, miembros y relaciones) junto con sus respectivas propiedades y posibles estereotipos a aplicar. A la hora de crear y/o modificar uno de los componentes, para facilitar la tarea de modelado, propone valores por defectos a sus componentes. Así también, reduce la posibilidad de modelados incorrectos ya que solamente permite establecer valores apropiados y estereotipos aplicables de acuerdo al componente que se esté modelando. Un creador de proyectos puede asignar a otros miembros para que puedan colaborar y manipular sus proyectos.

La arquitectura de la herramienta está basada en un esquema cliente-servidor, utilizando únicamente tecnologías Web de código abierto. Entre las tecnologías Web utilizadas podemos citar a: *KineticJS 4.5*, *ExtJS 4.2*, *Python Tornado 3.0*, *DjangoORM 1.5* y *Jinja2*.

En la figura 5 se observa un ejemplo de un modelado parcial de un proyecto. Se pueden observar gráficamente los módulos, los miembros con sus respectivos tipos de datos y las relaciones existentes entre los diversos miembros. En la

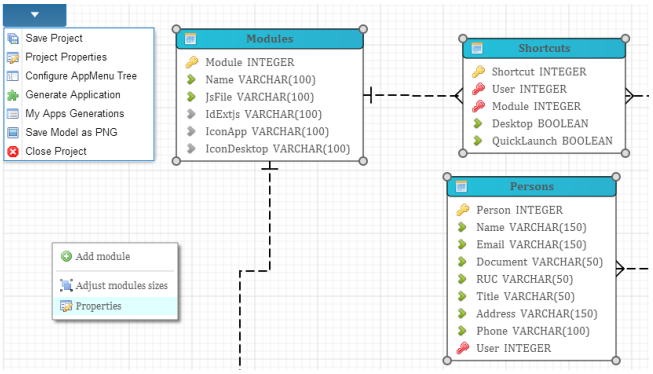

Figura 5: Ejemplo en YAACASE.

esquina superior izquierda se puede observar una lista de acciones posibles a realizar. Debajo del menú de acciones se observa un cuadro que es desplegado como reacción al evento *clic derecho* en el lienzo, y permite agregar un nuevo módulo, cambiar las propiedades del proyecto, entre otros.

En la medida en que se modelan las aplicaciones con *YAACASE*, estas pueden ser transformadas al código fuente de una aplicación funcional realizando una solicitud de generación con un determinado tipo de salida (o plataforma destino). Esta transformación es realizada por *YAALATOR*, de sus siglas inglés *Yet Another Agile transLATOR*. La transformación de modelo a código se encuentra basada en el reporte técnico presentado por Orban, en dónde propone utilizar el motor de traducción basado en plantillas denominado *Jinaj2* [11]. En el *Listing 1.1* se puede observar un ejemplo de aplicación de reglas de transformación para generar código SQL, en donde se van iterando en todos los módulos del proyecto para ir creando las tablas, y al mismo tiempo examinando algunos estereotipos para producir el tipo de dato correcto para la columna dada.

```
1 {% for module in project.modules %}
2 CREATE TABLE IF NOT EXISTS `{{ toLower(project.name) }}`.`{{
      toLower(module.name) }}` (
3 {% for member in module.members %}
4   {% if member.type == "INTEGER" %} `{{ member.name }}`
        INT({{ member.length }}) {% endif %}
5   {% if not member.allowNull %} NOT NULL {% endif %}
6   {% if member.defaultValue != '' %} DEFAULT '{{
        member.defaultValue }}' {% endif %}
7 );
8 {% endfor %} {% endfor %}
```

Listing 1.1: Ejemplo de regla de transformación.

YAALATOR se encarga de cargar todas las plantillas de traducción y los archivos de configuración que guían la traducción, para que posteriormente pueda

recibir un modelado UML y enviarlo al motor de traducción *Jinja2*, el cual se encargará de aplicar dicho modelo a las plantillas y generar así el código fuente. El archivo de configuración que guía al traductor debe ser construido para cada una de las posibles plataformas destino de generación. Dicho archivo tiene el formato *JSON* y está compuesto por campos bien definidos.

3. Validación de la propuesta YAAMDDA

Con el fin de poder evaluar y validar la propuesta YAAMDDA, se procedió a desarrollar un caso de estudio comparativo entre dos enfoques AMDD en proyectos de desarrollo de sistemas informáticos: *AMDD Manual* y *CASE Ágil usando la propuesta YAAMDDA*. El motivo por el cual se decidió realizar el caso de estudio entre estos dos enfoques AMDD fue debido a que, como se menciona en la introducción, actualmente en los proyectos que siguen el enfoque AMDD tan solo el 20 % de ellos son desarrollados siguiendo el enfoque CASE Ágil en contraste con el AMDD Manual en donde se estima un 70 %-80 % [10]. Dicho esto, fue muy motivador el poder contrastar estos enfoques en casos de estudio y determinar si con el enfoque AMDD CASE Ágil se pudiere obtener resultados prometedores o mejoras en comparación con el AMDD Manual.

La estructura, el esquema y las recomendaciones utilizadas en el caso de estudio se encuentran totalmente basadas en el framework propuesto por Runeson et al. [12]. El caso de estudio ofrece una novedosa contribución al estado del arte en lo que se refiere a la aplicación y comparación de dos enfoques distintos de desarrollo dirigido por modelos ágil para desarrollar RIAs Enterprise. La hipótesis especificada previo al comienzo del caso de estudio se encuentra definida como: la productividad de un equipo de desarrollo de RIAs Enterprise siguiendo la propuesta *YAAMDDA (AMDD ágil CASE)* es superior a la de un equipo de desarrollo que sigue el enfoque *AMDD Manual*. El caso de estudio realizado se encuentra categorizado según Runeson et al. [12] como: *Caso de estudio integrado singular (Embedded Single-Case Study)*. El caso de estudio es singular debido a la existencia de un único contexto bajo el cual se realiza el análisis y es integrado puesto que dicho contexto hay dos unidades de análisis.

El contexto del caso de estudio se encuentra basado en el desarrollo de un sistema que permite llevar un control de las llegadas y salidas de empleados y profesores de una institución académica. Dicho contexto contempla dos unidades de análisis, puesto que se han conformado dos equipos de desarrollo de 3 personas encargadas de construir la aplicación, en donde cada equipo utilizó un enfoque AMDD distinto. Todos los equipos de desarrollo fueron integrados por alumnos del último año de la carrera de Ingeniería Informática de la Universidad Católica "Nuestra Señora de la Asunción". Un aspecto importante a destacar es que el cliente encargado de brindar los requerimientos a los desarrolladores carecía de conocimientos avanzados de tecnologías de la información. La principal fuente de información consistió en los *archivos de datos*, mediante los cuales se

pudieron recolectar datos cuantitativos. Se realizaron además *entrevistas* a los equipos de desarrollo al final de cada iteración, con la intención de obtener datos cualitativos. Las métricas utilizadas para el caso de estudio fueron obtenidas de Ambler et al. [10], y son las siguientes: *Velocidad*, *Tiempo de actividad (Activity time)*, *Densidad (Density)*, *Tiempo invertido (Time invested)*. Por motivos de confidencialidad y con el fin de brindar el anonimato pactado con las personas de los equipos de desarrollo, los grupos fueron renombrados de la siguiente manera:

- Enfoque CASE Ágil (YAAMDDA): Equipo A1 - Miembros: A11, A12, A13.
- Enfoque AMDD Manual: Equipo A2 - Miembros: A21, A22, A23.

Se realizaron análisis de datos cuantitativos y cualitativos, destacando que los mismos no pueden ser generalizados, pero que sí brindan una idea general acerca de la hipótesis fundamental que se desea corroborar mediante el caso de estudio. Inicialmente se presenta la *densidad* de los proyectos, mediante el análisis de puntos función utilizando el estándar de la *IFPUG* [13] y de líneas de código *CLOCK*[1], con el fin de tener una idea de la complejidad en tamaño del proyecto.

Cuadro 1: Densidad de los sistemas desarrollados

	Métrica	**Automáticamente**	**Manual**	Σ
A1	Puntos Función	276	24	300
	Líneas de Código	24223	667	24890
	Porcentaje LDC	97.30 %	2.70 %	100 %
A2	Puntos Función	0	121	121
	Líneas de Código	0	3030	3030
	Porcentaje LDC	0 %	100 %	100 %

En el Cuadro 1 se pueden destacar los siguientes aportes:

- El tamaño del sistema desarrollado siguiendo el enfoque YAAMDDA es aproximadamente 3 veces más grande en términos de puntos función. Lo cual aparentemente es un punto negativo, puesto que si ambos sistemas cumplen los requisitos, al ser más grande (en puntos función y/o líneas de código), el mantenimiento posterior tenderá a ser más complicado. Cabe resaltar que en gran medida la causa de que el tamaño de las aplicaciones generadas con YAAMDDA sean superiores se debió a que eran más completas, de mejor apariencia, y uso más simple, características que fueron notadas a simple vista por los participantes del caso de estudio.
- Los equipos que siguen el enfoque YAAMDDA codifican menos líneas de código que los del enfoque AMDD Manual. Los del enfoque YAAMDDA

[1] Count Lines of Code (CLOC): `http://cloc.sourceforge.net/`

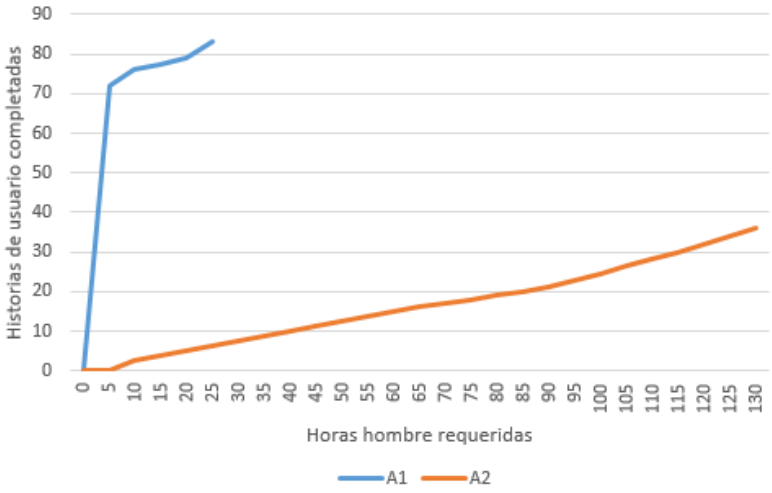

Figura 6: Historias de usuario completadas por horas hombre.

solamente se encargan de codificar aproximadamente el 3 % del total del sistema, ya que el resto es generado automáticamente.

En la figura 6 se puede observar el *tiempo invertido* por los equipos para desarrollar los sistemas. El tiempo invertido por el equipo A1 para desarrollar el sistema fue de 24 horas hombre, a diferencia de las 132 horas hombre invertidas por el equipo A2 para desarrollar el mismo sistema que el equipo A1. Por tanto, el sistema fue desarrollado aproximadamente 5.5x veces más rápido por el equipo que utilizó el enfoque YAAMDDA. Un aspecto muy importante a enfatizar es que el sistema desarrollado por el quipo A1 tiene aproximadamente 2.5x puntos función más que la del equipo A2 y 8.2x más líneas de código, y aún así el quipo A1 culminó el sistema 5.5x veces más rápido.

Con respecto al análisis cualitativo, uno de los aspectos más notorios fue que únicamente el equipo A1 pudo presentar prototipos funcionales ya desde el término de la primera iteración, por lo que los clientes iban interactuando y validando el trabajo realizado en la medida en que se desarrollaban las aplicaciones correspondientes. Cabe mencionar que el equipo A2 presentó recién un prototipo funcional al término de la tercera iteración. Cabe destacar que ambos equipos de desarrollo tuvieron en total 4 iteraciones para culminar el desarrollo del sistema.

Los siguientes puntos a destacar para el análisis cualitativo están basados en los comentarios de los desarrolladores y los clientes, que fueron surgiendo durante las reuniones realizadas a lo largo de todo el caso de estudio. Los miembros del equipo A1 destacaron que efectivamente ahorra mucho tiempo de trabajo a la hora de desarrollar específicamente Aplicaciones Enterprise. Al término del proyecto, por más de que nunca habían programado para la plataforma generada, destacaron que la generación de código proporciona un esquema base y bien ordenado que facilitó de gran manera el aprendizaje. Así también, mencionaron

que si hubiesen tenido que aprender por su cuenta, hubiesen tardado fácilmente tres a cuatro veces más de tiempo.

Respecto a la herramienta YAACASE, destacaron que fue muy simple y agradable de utilizar, puesto que sólo necesitaron de un navegador web para poder realizar el modelado, sin necesidad de instalar ninguna aplicación adicional. Además, se ha valorado positivamente el hecho de que sólo necesitaron realizar un único diagrama, muy similar al de un diagrama de base de datos físico con el cual ya tenían experiencia. Así también mencionaron que les pareció muy interesante el *prototipado veloz*, ya que una vez terminado el modelado, se podía generar la aplicación en cuestión de segundos y la misma era ya utilizable por los usuarios.

Con estos resultados podemos notar que para un área de dominio específico (como es el caso de las RIAs WEB-EAS), un enfoque soportado por herramientas CASE que faciliten la generación automática puede ser bastante ventajoso. Sin embargo, no sería posible generalizar el caso a otros dominios.

4. Trabajos Relacionados

Las propuestas metodológicas tradicionales para el desarrollo Web normalmente no siguen el enfoque AMDD. Sin embargo, existen algunos trabajos interesantes que normalmente se vinculan a proyectos. Los elementos tenidos en cuenta para seleccionar los proyectos actuales son: que soporten herramientas, que estén orientados a tecnologías actuales, específicamente WEB-EAS con tecnologías RIAs, y que contemplen aspectos metodológicos ya sea en forma explícita o implícita. Se han identificado una serie de criterios, para un análisis comparativo de los proyectos, que han sido clasificado de la siguiente manera: generales, orientados al diseño, orientados al proceso de transformación, y orientados a la tecnología de implementación. El siguiente cuadro presenta un resumen de los indicadores analizados incluyendo también la propuesta YAAMDDA.

Los resultados de este análisis ponen en evidencia que ninguno de los proyectos relacionados contemplan los criterios establecidos (en cuanto a diseño, transformación y tecnología de implementación) al mismo tiempo, salvo YAAMDDA. En el caso de diagramas requeridos, YAAMDDA es el único que contempla un único modelo. Cabe destacar, que esto es posible debido a que estamos orientando la propuesta a un dominio específico. Por otro lado, de todas las propuestas, es la única que utiliza una notación estándar como UML para modelar.

En lo que respecta al proceso de transformación, se puede notar que solo YAAMDDA sigue este enfoque. Brambilla et al. [14] mencionan dos métodos para realizar la traducción de modelo a código: el método basado en lenguajes de programación y el método basado en plantillas; y finalmente destacan que el método basado en plantillas brinda una flexibilidad ideal, permitiendo cambiar el texto a ser generado mediante la modificación de las plantillas sin necesidad de recompilar el código fuente del traductor. Finalmente, en cuanto a la tecnología de la

Criterios	XEO	AppFlower	Appcelerator	Cappucino	YAAMDDA
Generales					
Costo	Community: Gratuito Enterprise: No disponible	Community: Gratuito Premium: 50$ / mes	Community: Gratuito Enterprise: No disponible	Gratuito	Gratuito
Licencia	Community: GPL v3.0 Enterprise: Pago	MIT Licence	Apache Public Licence v2	LGPL v2.1	GPL v3.0
Basados en el diseño					
Cantidad de modelos requeridos	Requiere el modelado del diagrama de entidades, lógica y vista.	Requiere el modelado del diagrama de entidades, y de la vista de cada entidad.	Requiere el modelado del diagrama de entidades, lógica y vista.	Requiere el modelado del diagrama de entidades y vista.	Un único modelo UML
Patrones HCI	Aplicaciones generadas siguen diseño HCI	Aplicaciones generadas siguen diseño HCI	Aplicaciones generadas siguen diseño HCI	Aplicaciones generadas siguen diseño HCI	Aplicaciones generadas siguen diseño HCI
Modelado UML	No utiliza UML para modelar. Definen su propio método.	No utiliza UML para modelar. Definen su propio método.	No utiliza UML para modelar. Definen su propio método.	No utiliza UML para modelar. Definen su propio método.	Utiliza UML
Basados en el proceso de transformación					
Traducción basada en plantillas	No soporta la adición de nuevos templates de traducción.	No se pueden agregar nuevos templates. Todo se almacena en su plataforma en la nube.	No soporta la adición de nuevos templates de traducción.	No soporta la adición de nuevos templates de traducción.	Soporta plantillas y extensión de las mismas
Plantaformas de generación	JavaEE Server, ExtJS Web Client, MySQL, PostgreSQL.	PHP Cloud Server, ExtJS Web dinámico, MySQL, PostgreSQL.	PHP-Python-Ruby Servers, MySQL, PostgreSQL, Cliente Web Dinámico JS HTML5 CSS	PHP-Python-Ruby Servers, MySQL, PostgreSQL, Cliente Web Dinámico Objective-J	Python-Django-Java-Spring Servers, MySQL, PostgreSQL, ExtJS Web dinámico, JS HTML5 CSS
Personalización de la GUI requerida	Muy personalizable, pero se requiere un sistema extra para el diseño de la interfaz.	No muy personalizable, y además requiere de mucha intervención del modelador.	Muy personalizable, pero se requiere un sistema extra para el diseño.	Muy personalizable, pero se requiere un sistema extra para el diseño de la interfaz.	Muy pesonalizable, pero se requieren plantillas de generación distintas
Basados en la tecnología de la implementación					
Servicios Web	REST y SOAP solo presente en la versión enterprise (pagada).	Con soporte para generación de aplicaciones web RESTful.	Con soporte para generación de aplicaciones web RESTful.	Utiliza SOAP. Agregar un plugin para soporte RESTful.	Con soporte para generación de aplicaciones web RESTful.
Generación de reportes PDF	Solo reportes en formato HTML.	Solo reportes en formato HTML.	Solo reportes en formato HTML.	Solo reportes en formato HTML.	Reportes en formato HTML y PDF.
Modelado usando interfaz Web	Sin soporte. Requiere un plugin para Eclipse.	Soporta modelado desde navegador web.	Sin soporte. Requiere de varias aplicaciones extras.	Sin soporte. Requiere una aplicación extra.	Soporta modelado desde navegador web.

Figura 7: Comparación de proyectos

implementación, la figura 7 nos muestra que YAAMDDA cumple con todos los requisitos definidos, plataformas destino, formatos de reportes, acceso vía Web, entre otros.

5. Conclusión

Se pudo constatar la fortaleza de varias herramientas existentes en el mercado, que fueron desarrolladas con el fin de agilizar el desarrollo de Aplicaciones WEB-EAS, pero al mismo tiempo se verificó que existían varios puntos considerados muy importantes que no fueron totalmente contemplados por las mismas. Algunas de las más importantes a citar son: permitir el modelado web, evitar requerir múltiples modelos, y aplicaciones Web basadas en tecnologías estandarizadas, entre otros.

YAAMDDA surge para dar un soporte computacional al enfoque *AMDD CASE Ágil* con la intención de proporcionar una alternativa a las problemáticas mencionadas. *YAAMDDA* brinda un lenguaje, un proceso, un traductor de modelo a código y una herramienta de modelado, que están orientadas a agilizar el desarrollo de las Aplicaciones WEB-EAS, en particular las RIAs.

Cabe destacar que si bien los resultados del caso de estudio constituyen solo una primera validación, aplicando la propuesta CASE Ágil *YAAMDDA*, se obtuvieron resultados interesantes y alentadores. En particular *YAAMDDA* aumentó la

productividad ya que requirió 5.5x veces menos horas hombre y permitió implementar un numero definitivamente mayores de historias de usuario, comparando con el equipo de desarrollo que siguió el enfoque AMDD Manual.

Entre los trabajos futuros más relevantes podemos citar la necesidad de realizar ajustes sobre la herramienta CASE y desarrollar plantillas de traducción para otras plataformas. Además, para poder generalizar los resultados obtenidos serán necesarios otros casos de estudios y experimentos formales.

Referencias

1. M. Fowler, *Patterns of Enterprise Application Architecture*, 1st ed. Addison-Wesley, 2003.
2. M. Busch and N. Koch, "Rich internet applications, state of the art," Ludwig-Maximilians-Universitat Munchen, Germany, Institute for Informatics, Tech. Rep., December 2009.
3. S. Meliá, J. Gómez, S. Pérez, and O. Díaz, "A model-driven development for gwt-based rich internet applications with ooh4ria," in *ICWE*, 2008, pp. 13–23.
4. Z. Mansor, S. Yahya, and N. Arshad, "Software development life cycle agile vs traditional approaches," in *International Journal on New Computer Architectures and Their Applications (IJNCAA)*, vol. 1, no. 3, 2011, pp. 942–952.
5. M. Lapham, S. Miller, B. Hackemack, and A. Schenker, "Agile methods: Selected dod management and acquisition concerns," Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, EEUU, Tech. Rep., October 2011, cMU/SEI-2011-TN-002.
6. Y. Leau, W. Loo, W. Tham, and S. Tan, "Software development life cycle agile vs traditional approaches," in *International Conference on Information and Network Technology (ICINT)*, vol. 37. LACSIT Press, 2012, pp. 162–167.
7. T. Kapteijns, S. Jansen, S. Brinkkemper, and R. Barendse, "A comparative case study of model driven development vs traditional development the tortoise or the hare," in *4th European Workshop on "From code centric to model centric software engineering: Practices, Implications and ROI"*, vol. 4, 2009, pp. 22–33.
8. K. Krogmann and S. Becker, "A case study on model-driven and conventional software development: The palladio editor," in *Software Engineering*, W.-G. Bleek, H. Schwentner, and H. Züllighoven, Eds., vol. 106. GI, 2007, pp. 169–176.
9. P. Parviainen, J. Takalo, and S. Teppola, *Model-Driven Development: Processes and practices*, 1st ed. VTT Technical Research Center of Finland, 2009.
10. M. Lines and S. Ambler, *Introduction to Disciplined Agile Delivery*, 1st ed. IBM Press, May 2012, pp. 50–55.
11. D. Orban, "Templating and automatic code generation for performance with python," Groupe d'études et de recherche en analyse des décisions and the Department of Mathematics and Industrial Engineering, École Polytechnique, Montréal, QC, Canada., Tech. Rep., 2011.
12. P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering*, 1st ed. John Wiley and Sons, 2012.
13. M. Bradley *et al.*, "Function point counting practices manual, release 4.1," Chairperson Counting Practices Committee, Westerville, Ohio, EEUU, Tech. Rep., 1999.
14. M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 1st ed. Morgan and Claypool Publishers, 2012.