La Arquitectura de Software en el Proceso de Desarrollo: Integrando MDA al Ciclo de Vida en Espiral

Valeria S. Meaurio, Eric Schmieder
Escuela de Posgrado, Facultad Regional Buenos Aires, Universidad Tecnológica Nacional
Ciudad Autónoma de Buenos Aires, Argentina
valeriameaurio@hotmail.com, ericschmieder@gmail.com

***Resumen* — En la actualidad el desarrollo de software debe enfrentarse a una serie de problemas, por ejemplo, la rapidez con que el software debe estar disponible para su uso, el alto grado de evolución de los sistemas actuales y el nivel de complejidad de los mismos, entre otros. Contar con un modelo de la arquitectura en etapas tempranas se hace evidente, puesto que anticiparnos a la especificación detallada del sistema nos permite contar con un modelo de alto nivel de la alternativa de solución a los requerimientos planteados, que en sucesivos refinamientos conducirán al producto final. En vistas de estos problemas la propuesta de la OMG arquitectura dirigida por modelos (Model Driven Architecture - MDA) propone la construcción de software como la trasformación sucesiva de modelos, desde un alto nivel de abstracción hasta el nivel de implementación en una plataforma concreta. El objetivo del presente trabajo propone aplicar el modelo MDA dentro del ciclo de vida en espiral propuesto por Bohem a fin de establecer el impacto de la arquitectura de software en las distintas etapas de este ciclo de vida y como MDA minimiza el riesgo.**

***Palabras Clave* — MDA, Arquitectura de Software.**

I. Introducción

La arquitectura dirigida por modelos (Model Driven Architecture – MDA) promovida por la OMG propone que los modelos en sus diferentes niveles de abstracción conducen todo el proceso de desarrollo de software, desde los modelos independientes de la plataforma (CIM y PIM) hasta los modelos dependientes de la plataforma (PSM) y la generación automática de código a partir de los mismos. Para llevar adelante este pasaje entre modelos propone mecanismos de transformación[1].

El modelo en espiral es un modelo evolutivo que permite el desarrollo rápido de versiones incrementales del software y a diferencia de otros paradigmas incorpora un nuevo elemento: el análisis de riesgo. Este modelo, representado mediante un espiral, define cuatro actividades principales 1) planificación, 2) análisis de riesgo, 3) ingeniería y 4) evaluación del cliente[2].

En este trabajo son dos los objetivos primordiales a cubrir. El primero es identificar dentro del ciclo de vida en espiral el grado en que participa la arquitectura del software en las distintas fases del mismo y el segundo aplicar el modelo MDA dentro del ciclo de vida en espiral evaluando el impacto de MDA sobre los riesgos.

El resto del artículo está organizado de la siguiente manera: en el apartado 2 se presenta el modelo MDA sus características, ventajas y desventajas; en el apartado 3 se detalla el ciclo de vida de desarrollo en espiral; en el apartado 4 se analiza el impacto de la arquitectura de software en el ciclo de vida de desarrollo en espiral; en el apartado 5 se propone una alternativa para la implementación de MDA en el ciclo de vida en espiral; y en el apartado 6 las conclusiones y posibles líneas de trabajo futuro.

II. Modelo MDA

MDA es el acrónico de Model Driven Architecture promovido por la OMG que propone basar el desarrollo del software en modelos especificados utilizando UML para que a partir de esos modelos se realicen transformaciones que generen código u otro modelo con menor nivel de abstracción[3].

MDA define un marco para procesar y relacionar modelos, distinguiendo los siguientes modelos:

- CIM (Computation Independent Model): se centra en los requerimientos y representa el nivel más alto del modelo de negocio. Es un modelo de dominio.
- PIM (Platform Independent Model): es un modelo de alto nivel del sistema independiente de cualquier tecnología o plataforma. Permite una abstracción de las características técnicas.
- PSM (Platform Specific Model): Describe el sistema de acuerdo con una tecnología de implementación determinada. Proporciona independencia entre la capa de negocio y la tecnología empleada.
- ISM (Implementation Specific Model): La generación de código se realiza automáticamente a partir de cada PSM.
- MDA – Cartridges: un cartucho MDA contiene las reglas necesarias para realizar una transformación de modelos.

Entre las ventajas de este modelo se identifican las siguientes:

- Separación de responsabilidades: Separar la especificación de la funcionalidad del sistema de su implementación sobre una plataforma en particular
- Productividad: Controlar la evolución desde modelos abstractos a implementaciones, tendiendo a aumentar el grado de automatización.
- Portabilidad: Se logra enfocando el desarrollo en el PIM. Al ser un modelo independiente de cualquier tecnología, todo lo definido en el es totalmente portable.

- Mantenimiento y documentación: a partir del PIM se generan los PSM y a partir de estos el código. Básicamente el PIM desempeña el papel de la documentación de alto nivel. La trazabilidad está asegurada debido a que los cambios en el PIM se reflejan regenerando PSM y código.

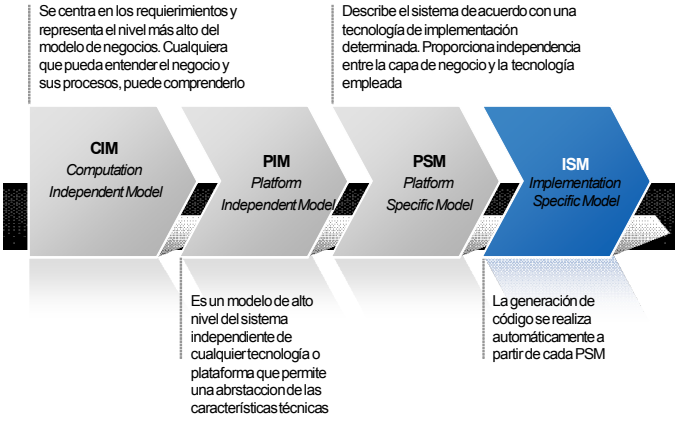

Fig.1 Modelos MDA

A modo de ejemplo en la figura 2 podemos visualizar en ciclo de vida iterativo implementado con MDA, donde en la parte izquierda visualizamos las distintas etapas del ciclo y a la derecha los artefactos de salida de los mismos.

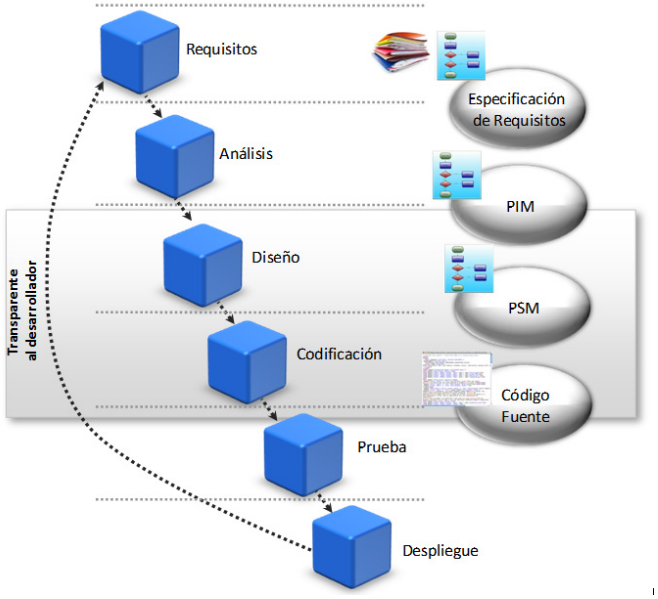

Fig. 2 - Proceso de desarrollo iterativo con MDA

Sintetizando los cambios en los procedimientos de negocio o la incorporación de nuevas funcionalidades puede hacerse de una manera más simple sin tener que hacer cambios en todos los niveles del proyecto. Simplemente se desarrollan los cambios en el PIM, y estos se reflejan en toda la aplicación, consiguiendo una disminución de trabajo en el equipo de desarrollo, una reducción en los costos del proyecto y por ende se aumenta la productividad [4].

MDA se apoya sobre los siguientes estándares para llevar a cabo su función:

- UML: lenguaje de modelado adoptado por MDA, empleado para la definición de los PIM y PSM. Cabe mencionar que los modelos de clases son los más importantes dentro de MDA.
- MOF (Meta Object Facility): establece un marco común de trabajo para las especificaciones de OMG, a la vez que provee un repositorio de modelos y meta modelos. Mediante MOF puede definirse cualquier lenguaje de modelado incluyendo UML.
- XMI (XML Metadata Interchange): define una traza que permite transformar modelo UML en XML para poder ser tratados automáticamente por otras aplicaciones.
- CWM (Common Warehouse Metamodel): Define un formato común de intercambio para metadatos en data-warehouse, también provee un lenguaje común y definiciones de meta modelos para datos. El meta modelo tiene mucho en común con el meta modelo UML y agrega meta clases para, por ejemplo, modelar bases de datos relacionales.

III. CICLO DE VIDA DE DESARROLLO EN ESPIRAL

Este modelo (Barry Boehm 1986), incorpora métodos de proceso que están influenciados por el control y gestión del riesgo para el análisis y estructuración del proceso de desarrollo. Se encuentra representado por ciclos de desarrollo evolutivo e iterativo en forma de espiral, cuyo avance angular representa el progreso del desarrollo, en tanto que el desplazamiento radial desde el centro hacia fuera indica el incremento de los costos de desarrollo en forma acumulativa [5].

La Figura 3 permite visualizar gráficamente el comportamiento de este tipo de ciclo de vida, en ella se pueden distinguir en sus cuatro cuadrantes las cuatro actividades principales del modelo [6]:

- Determinación de Objetivos y Alternativas: Involucra la determinación de objetivos del proyecto, alternativas y restricciones, y, en etapas posteriores, la valoración de los resultados de las tareas de ingeniería previas.
- Análisis de riesgo: Estas actividades permiten gestionar los riesgos asociados al proceso de desarrollo. Se materializan en el análisis de alternativas e identificación y resolución de los riesgos que puedan hacer fracasar el proyecto o sobrepasar el presupuesto o plazo fijado. Producido el análisis de riesgo, se toma la decisión de continuar o no con el desarrollo.
- Ingeniería: Abarca las actividades inherentes al desarrollo del producto y comprende la construcción de los prototipos de nivel cada vez mas refinados, a medida que se produce la evolución del sistema, hasta llegar al producto final.
- Planificación: Involucra todo lo atinente a las tareas de planificación y estimaciones del proyecto en las distintas etapas.

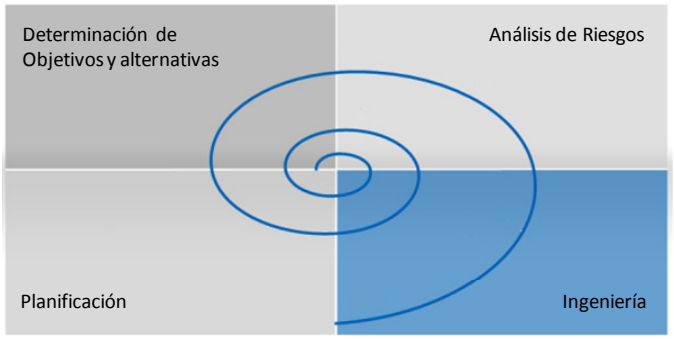

Fig.3 Modelo en Espiral

IV. Impacto de la Arquitectura de Software en el Ciclo de Vida de Desarrollo en Espiral

La Arquitectura de Software (AS) representa un alto nivel de abstracción que permite que la mayoría de los participantes, puedan utilizarla como base para crear entendimiento mutuo, formar consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales. La AS representa la encarnación de las decisiones de diseño más tempranas sobre un sistema. La evaluación crítica de una AS conduce a una comprensión más clara de los requerimientos, las estrategias de implementación y los riesgos potenciales [7].

Para ubicar a estas tareas dentro del proceso de desarrollo de software se usa al modelo de espiral propuesto por Boehm. A través de él se describen las fases y las tareas que le corresponden a la arquitectura de software, como se ilustra en la Figura 4.

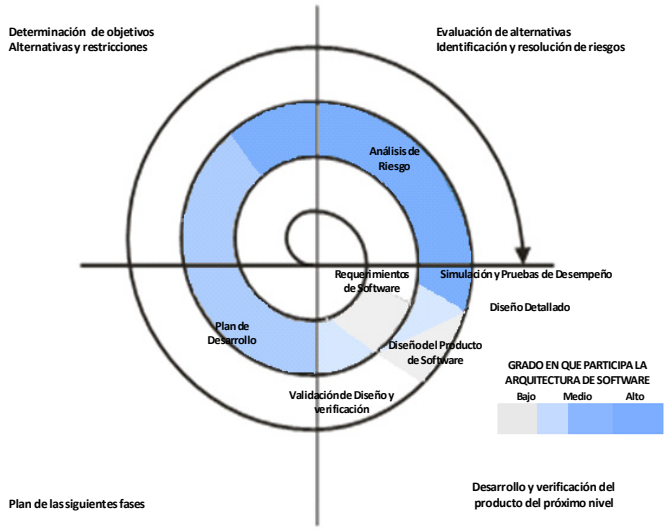

Fig. 4 – La arquitectura de software en el ciclo de vida de software del modelo en espiral de Bohem

Una vuelta de espiral que cubre a estas cuatro fases es un ciclo, indica el nivel de desarrollo que se tiene del sistema. La intervención de la AS dentro de este proceso está marcada con la parte sombreada de la Figura 3 La intensidad del sombreado indica el grado de intensidad de las tareas arquitectónicas y el grado en que ésta se involucra en el proceso de desarrollo del software.

La AS empieza a ser formada desde que inicia el ciclo de la especificación de los requerimientos de software, cuando se establecen los puntos de referencias del dominio del problema para su entendimiento el arquitecto debe interpretar, conciliar y proponer alternativas de solución en función del conocimiento de los expertos del dominio y de los planificadores del proyecto tal como señala [8] que ilustra esquemáticamente la interacción entre estos actores, mostrado en la Figura 5. El propósito de esta interacción es llegar a establecer una integridad conceptual del modelo a desarrollar expresada mediante la arquitectura.

"La integridad conceptual es la clave para el diseño de sistemas con éxito y tal integridad conceptual puede ser tenida por un pequeño número de mentes llegando juntos a diseñar la arquitectura del sistema" [9]

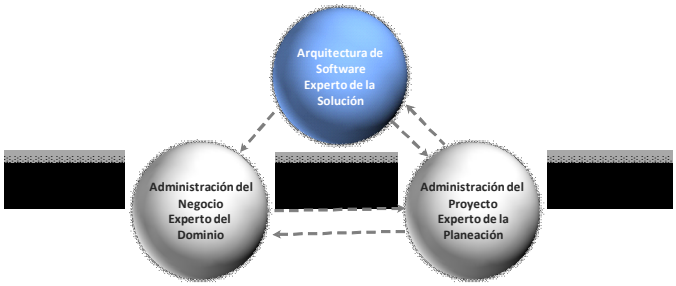

Fig. 5 - Interacción de la AS con los actores involucrados en el ciclo de requerimientos. (Tomado de (McBride 2007).

Al avanzar al siguiente ciclo donde se realiza el desarrollo del plan, se identifican los requerimientos significativos arquitectónicamente que son aquellos que tienen un impacto esencial sobre la arquitectura [10], esto consolida la integridad conceptual antes mencionada y refina la primera arquitectura que expresa principalmente a elementos genéricos que representan a los requerimientos funcionales.

En el siguiente ciclo correspondiente a la fase de determinación de objetivos, alternativas y restricciones, se toman decisiones arquitectónicas que involucran a los actores del diseño del sistema, decisiones como las que se citan en [11] que tienen que ver con cuestiones de cómo implementar los requisitos y con situaciones de riesgos, costos y viabilidades de las posibles alternativas que se tienen en la organización dónde el sistema se desarrolla, por ejemplo si se usa un sistema legado, si se construye uno nuevo, o se usan elementos(componentes) de software prefabricados disponibles en el mercado. Conforme se avanza por este ciclo los requerimientos se van incorporando en forma iterativa a la arquitectura hilándose entre si, como se describe en [12], donde se presenta una adaptación del modelo de Boehm destacando cómo los requisitos están al mismo nivel que la arquitectura enfatizando la igualdad en importancia entre la arquitectura y los requisitos, en el sentido de que un entendimiento oportuno de los requerimientos y una elección adecuada de la arquitectura son claves para la gestión de proyectos.

Los requisitos que son derivados de las necesidades de los usuarios son conocidos como funcionales o también como características operacionales del sistema, y son útiles para formar los elementos básicos de cada estructura que integra la AS, mientras que los requisitos que capturan la mayoría de las facetas de cómo estos requisitos funcionales deben llevarse a cabo se les conoce como requisitos no funcionales, los cuales pueden ser expresados en términos de requisitos de calidad que tienen su origen en los servicios de calidad dentro del ámbito de las comunicaciones cómo se describe en [13].

Los requisitos de calidad están conformados por un conjunto de atributos de calidad, los más comunes de estos son escalabilidad, seguridad, desempeño y fiabilidad, estos sirven para dirigir el diseño de una AS, como se muestra en el método de diseño propuesto por Bass [14], llamado "diseño dirigido por atributos", conocido técnicamente por sus siglas en inglés como ADD (Attibute Driven Design). Mediante estos atributos de calidad se llega a refinar las estructuras diseñadas con los requisitos funcionales llegando a producir la especificación de la arquitectura, que constituye el producto más importante generado en este ciclo de desarrollo del software.

En el siguiente ciclo correspondiente al análisis de riesgos y elaboración de prototipos, la AS tiene una doble función, la primera está encaminada hacia ella misma en el sentido de que es analizada para comprobar si la AS soporta los

requerimientos de calidad planteados, y la segunda función es el usarla como un prototipo para analizar el comportamiento de algunos aspectos del sistema antes de proceder a su diseño.

En el siguiente ciclo donde se construye el diseño de los productos de software y se realiza la validación y la verificación del mismo, se ejecutan los prototipos de la AS construidos en la fase anterior y se hacen las adecuaciones pertinentes en caso necesario, después de esto la AS es usada para construir el diseño del sistema. El puente entre los elementos arquitectónicos y los de diseño pude realizarse mediante técnicas de transformación de modelos. En cualquier caso se debe establecer un medio para conservar la traza entre estos elementos de grano grueso, que son los elementos arquitectónicos, con los elementos de grano fino que corresponden al diseño para asegurar que la arquitectura sea quien gobierne de manera efectiva las siguientes tareas del proceso.

V. UNA ALTERNATIVA PARA LA IMPLEMENTACIÓN DE MDA EN EL CICLO DE VIDA EN ESPIRAL

Como mencionamos en los apartados anteriores el ciclo de vida de desarrollo en espiral propone un desarrollo iterativo e incremental donde la arquitectura guía el proceso de desarrollo y el riesgo es evaluado en cada iteración. Los artefactos producidos y consumidos entre las diferentes etapas están desconectados entre si y del código fuente que representan. A medida que los sistemas van siendo modificados, la distancia existente entre el código y los artefactos de modelado previamente construidos se incrementan.

La alternativa propuesta comienza obteniendo en las etapas iniciales una primera versión de la AS para lo cual se utiliza como estrategia la separación o descomposición.

La separación es una estrategia aplicada a la hora de definir la AS para reducir la complejidad inherente que el software representa, puesto que al separar los diferentes asuntos de interés que intervienen, se logra una mejor compresión de sus elementos y de las relaciones que se establecen entre ellos.

Existen varias maneras de realizar esta separación; se puede realizar una separación modular, dividiendo al sistema en funciones representadas por módulos que se relacionan entre sí de manera jerárquica o se puede descomponer al sistema en clases, donde cada clase representa una abstracción y estas se relacionan entre sí de diversas formas.

El proceso parte del dominio del problema de donde se obtiene la especificación de los requisitos, dicha especificación sirve de entrada para iniciar el proceso de descomposición.

Como resultado de este proceso de descomposición obtenemos los primeros elementos arquitectónicos.

Los arquitectos de software junto con los expertos en el dominio serán quienes tengan gran responsabilidad en el proceso de de obtener esta primera versión de la AS.

A partir de esta primera versión de la AS los responsables de la administración del proyecto identificaran cuales son los requisitos prioritarios para avanzar en el plan.

De acuerdo a las prioridades establecidas para los requisitos, se evalúan las distintas alternativas de solución, teniendo presentes las restricciones, los requisitos de calidad y el análisis de riesgo sobre cada una de estas y se avanza en el refinamiento de los mismos generando los PIMs de cada uno de estos.

El desarrollo del PIM se corresponderá casi en su totalidad con la fase de análisis en el desarrollo tradicional. Se desarrollará un modelo formal que especifique qué debe realizar la aplicación. Este modelo estará libre de detalles de implementación, pero será extremadamente detallado.

La especificación de cómo implementar el modelo del análisis (PIM) en una plataforma concreta, está dada por la definición de una Transformación.

Tomando como entrada a los PIM y las Transformaciones, se generan los PSM que a su vez pueden ser transformados en el código fuente de la aplicación. Ambas transformaciones se realizan de manera automática.

A partir de los resultados obtenidos, se valida con los Stakeholder y de ser necesario se realizan ajustes sobre los PIM. Estos ajustes se incorporaran a la planificación de la siguiente etapa, junto con los requisitos ya planificados para la misma.

El proceso continúa hasta que todos los requisitos fueron contemplados por una solución de software y la misma se encuentra validada, probada e implementada.

VI. CONCLUSIONES

En este artículo hemos presentado un marco de trabajo identificando el grado de participación de la arquitectura del software en las distintas fases del ciclo de vida de desarrollo en espiral y aplicando MDA en el mismo.

Una de las grandes deficiencias de los ciclos de vida tradicionales, está dada por la perdida en la trazabilidad entre los artefactos generados por las distintas etapas a medida que se avanza en el proyecto. Cambios en los requisitos, nuevos requisitos, modificaciones de diseño o errores detectados en etapas avanzadas, generalmente no se actualizan en los artefactos generados en etapas anteriores.

MDA al permitir la doble transformación automática PIM-PSM y PSM-código fuente hace que el esfuerzo se centre en los modelos de alto nivel y al regenerar el resto de los modelos, la trazabilidad está garantizada. Esto hace que las iteraciones dentro del ciclo de vida sean más eficaces para afrontar estos cambios, minimizando de esta manera los riegos asociados.

Otra de las ventajas está dada por la separación de los aspectos tecnológicos; si durante el proceso se produjeran cambios en los requisitos tecnológicos de la aplicación, esto implicaría un ajuste en las definiciones de las transformaciones a realizar, sin mayor impacto en los PIM reduciendo así los riegos de este tipo de modificaciones.

REFERENCIAS

[1] Pablo Amaya Barbosa, Carlos González Contreras, Juan M. Murillo Rodríguez. Aspect MDA: Hacia un desarrollo incremental consistente integrando MDA y Orientación a Aspectos, Paper, Universidad de Extremadura

[2] Lidia Fuentes, Pablo Sanchez. Desarrollo de software con aspectos dirigido por modelos, Paper, Universidad de Málaga

[3] Lic. Liliana Favre, Dr. Gustavo Rossi. Componentes MDA para patrones de diseño. Tesis, Universidad Nacional de La Plata

[4] Begoña Cristina Pelaya Garcia-Bustero. TALISMAN: Desarrollo ágil de software con arquitecturas dirigidas por modelos. Tesis Doctoral, Universidad de Oviedo.

[5] Roger S. Pressman. Ingenieria del Software Un Enfoque Practico, 3ra ed., Mc Graw Hill

[6] Lic. Claudio Rancan, Ing. Paola Britos. Gestión de Configuración de Productos de Software en etapa de desarrollo. Trabajo Final de especialización, ITBA.

[7] Rogelio Noé Limón Cordero. Las vistas arquitectónicas de software y sus correspondencias mediante la gestión de modelos, Tesis Doctoral, Universidad Politécnica de Valencia

[8] McBride Matthew R. The Software Architect, ACM Communications, Vol. 50, Num, 5, pp.75-81, 2007

[9] Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford. Documenting Software Architectures: Views and Beyond, Addison-Wesley Professional; 1st edition (September 26, 2002),isbn: 0201703726, // as tutorial it is 25th International Conference on Software Engineering (ICSE'03), IEEE, 2003.

[10] Jazayeri, M., Ran Alexander, Van der Linden F. Software Architecture for Product Families. Addison- Wesley ISBN-10 0201699672, 2000

[11] Jeff Tyree and Art Akerman, Architecture Decisions: Demystifying Architecture, IEEE Postmodern Software Design, March/April 2005 (Vol. 22, No. 2), 2005, pp 19-27

[12] Nuseibeh Bashar. Weaving Together Requirements and Architectures, IEEE, Computer, vol 34, n 3, 2001, pp. 115-119

[13] Manola Frank. Providing Systemic Properties (Ilities) andQuality of Service in Component-Based Systems, Report prepared by Object Services and Consulting, Advanced Information Technology Services Architecture, under contract DASW01 94 C 0054 for the Defense Advanced research Projects Agency, 1999. http://www.objs.com/aits/9901-iquos.html#character. último acceso: oct/2007

[14] Bachmann, F.; Bass, L.; Chastek, G.; Donohoe, P.& Peruzzi, F. The Architecture Based Design Method.CMU/SEI-2000-TR-001 ADA375851. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000

Valeria S. Meaurio. Es Licenciada en Análisis de Sistemas por la Universidad de Buenos Aires. Se desempeña como Consultor Senior en TI en servicios financieros en Ernst & Young. Sus áreas de interés, análisis y especificación de requerimientos, metodologías de software, análisis de sistemas.

Eric Schmieder. Es Programador Universitario en Informática por la Universidad Católica de Salta, Licenciado en Informática por la Universidad Católica de Salta, Especialista en Project Management por la Universidad Tecnológica Nacional. Es responsable del equipo de desarrollo para el área inversiones del Banco Supervielle. Sus áreas de interés son las metodologías de ingeniería de software, el análisis de sistemas, la ingeniería de requisitos y el modelado conceptual.