Definición de una Ontología para el Proceso de DSDM considerando Requisitos No-Funcionales

David Ameller y Xavier Franch

Grupo de Investigación GESSI, Universitat Politècnica de Catalunya (UPC)
UPC-Campus Nord, edificio Omega, 08034 Barcelona
{dameller, franch}@lsi.upc.edu

Resumen. La consideración de los requisitos no-funcionales (RNF) en el proceso de desarrollo de software dirigido por modelos (DSDM) impacta en la educción del estilo arquitectónico del sistema software resultante, así como en la selección de las tecnologías más apropiadas para su implementación. A partir de un análisis de dicho impacto, en este trabajo se enumeran, definen y relacionan los conceptos fundamentales que emergen en el tratamiento de los RNF en el proceso de DSDM, y se ilustra su rol en el proceso mediante un escenarios de uso.

Palabras clave: desarrollo de software dirigido por modelos, requisitos no-funcionales, arquitecturas del software.

1 Introducción

Típicamente, los requisitos establecidos sobre un sistema software se clasifican como funcionales o no-funcionales. Mientras que los requisitos funcionales nos indican el comportamiento que se espera del sistema, los requisitos no-funcionales (RNF) [1] nos indican las restricciones que éste debe cumplir y las características que se deben potenciar (p. ej., eficiencia, mantenibilidad, usabilidad, etc.).

Una lección aprendida en la ingeniería del software es que el desarrollo de software sin considerar los RNF desemboca en mayores costes tanto para el cliente como para el desarrollador [2, 3]. Aun así, sigue siendo frecuente soslayar este tipo de requisitos. Por ejemplo, en muchas ocasiones la arquitectura y las tecnologías utilizadas vienen prefijadas sin considerarlos siquiera. En esta situación, si no consideramos los RNF de usabilidad, en un proyecto que requiera facilidades para personas discapacitadas, podríamos vernos obligados a cambiar la tecnología usada en el desarrollo de la interfaz del usuario a mitad del proyecto. O bien, podríamos detectar de forma tardía que se requiere un tipo de interoperabilidad entre los usuarios del sistema que nos obligara a replantearnos la arquitectura prefijada en el momento de integrar los componentes de la solución. Desde esta perspectiva, puede decirse que la calidad del software es el "grado en el que un conjunto de características inherentes cumple con los requisitos" [4]. Todo proceso de desarrollo de software debe tener en cuenta esta máxima y el desarrollo de software dirigido por modelos (DSDM) no es una excepción.

En el seno del grupo GESSI (http://www.lsi.upc.edu/~gessi) de la UPC estamos desarrollando ArchiTech, un marco de trabajo para DSDM en el que se consideran los RNF para: 1) educir (del inglés "elicit") los estilos arquitectónicos y tecnológicos más adecuados para un sistema software en desarrollo, y 2) posteriormente construir la arquitectura software resultante y seleccionar el conjunto de tecnologías correspondientes a dichos estilos que más se adecuen a las necesidades especificadas e implícitas de dicho sistema.

El primer problema que hemos encontrado es la falta de una ontología que introduzca y relacione todos los conceptos exigidos por ArchiTech. Por ello nuestra prioridad en esta fase de investigación es identificar, definir y relacionar con precisión los conceptos utilizados en ArchiTech (p. ej., tecnología, estilo arquitectónico, etc.).

En este artículo proponemos una primera solución al problema identificado mediante la construcción de una ontología para el proceso de DSDM basado en NFR tal y como se plantea en ArchiTech. Para ello, en la Sección 2 introducimos las ideas básicas de Architech, en la Sección 3 presentamos la ontología, en la Sección 4 ilustramos ambos mediante un escenario de uso, y en la Sección 5 resumimos las conclusiones e identificamos el trabajo futuro.

2 Impacto de los RNF en el DSDM

El DSDM es una forma de desarrollo de software que potencia los beneficios de partir de un modelo muy abstracto, el CIM, e ir bajando este nivel de abstracción paulatinamente pasando por el PIM, posteriormente el PSM y finalmente el código. Actualmente, este proceso de refinamiento semiautomático está centrado en los aspectos funcionales del sistema software, salvo unas pocas excepciones como [5, 6]. Así, los RNF se consideran y se toman decisiones a partir de ellos incluso cuando no forman parte del proceso DSDM, pero de la misma forma en que se han considerado normalmente durante las últimas décadas, es decir como una pieza de documentación o como un conocimiento implícito del desarrollador.

Nuestra meta en ArchiTech es colocar los RNF al mismo nivel que los requisitos funcionales dentro del DSDM. Ello implica que podamos modelarlos y que podamos trabajar con ellos de forma semiautomática. La diferencia principal en comparación con los requisitos funcionales, es que los RNF juegan un papel mayoritariamente decisional dentro del proceso DSDM.

La primera observación realizada sobre los RNF está centrada en su nivel de abstracción. Los RNF afectan de una forma u otra al proceso de DSDM dependiendo de su nivel de abstracción. Así, podemos diferenciar entre RNF con consecuencias en la arquitectura (NRF arquitectónicos) o en la tecnología (NFR tecnológicos), dicho de otra forma, requisitos que condicionan la construcción del PIM y requisitos que condicionan la construcción del PSM[1]. Existen otras clasificaciones de RNF para el proceso DSDM como la propuesta por V. Cortellessa en [5] (extendida en [6]), pero éstas no se adaptan a las necesidades de ArchiTech. Comentaremos las diferencias con esta propuesta en las conclusiones del artículo.

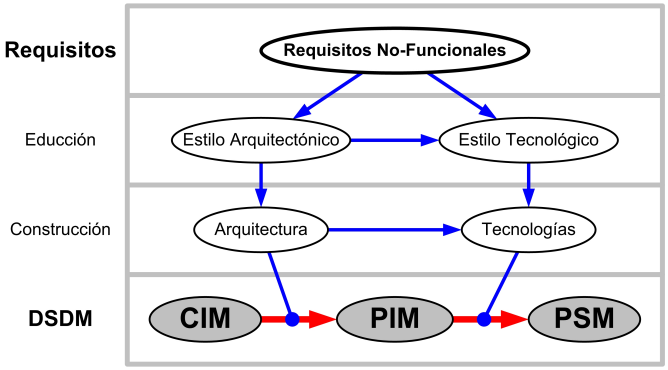

Fig. 1: Impacto de los RNF en el DSDM

[1] Usando el mismo razonamiento se podría considerar un nivel más abstracto de RNF que condicionaría la construcción del CIM, pero éste sale de los objetivos del presente artículo.

La forma en que los RNF arquitectónicos y tecnológicos afectan al proceso de DSDM es: en primer lugar nos ayudan a educir cuáles son los estilos que más favorecen a los RNF especificados y en segundo lugar nos dirigen el proceso de construcción de la arquitectura software y la selección de las tecnologías adecuadas para la implementación (v. Fig. 1).

3 Ontología para la Integración de los RNF en el DSDM

Esta ontología sigue los principios básicos descritos en [7]. Dividimos el estudio de la ontología en cuatro grandes grupos: requisitos, estilos, artefactos y modelos. Para todos ellos, definimos los términos básicos y mostramos sus relaciones mediante diagramas de clase UML. Para facilitar la lectura, los términos se numeran en orden de definición, y ese número *n* se usará como referencia a lo largo del artículo (como δ-*n*). Los términos que son comunes en la ingeniería del software se entrecomillarán con la definición que se considera más adecuada para la ontología (por ello aparecen en inglés). La ontología en una pieza puede consultarse en el anexo de este artículo.

Requisitos. Además de la ya mencionada distinción entre requisitos funcionales y requisitos no-funcionales, clasificándose estos últimos entre RNF arquitectónicos y RNF tecnológicos (v. Fig. 2).

1. *Requisito funcional*: "A system/software requirement that specifies a function that a system/software system or system/software component must be capable of performing." [8].
2. *Requisito no-funcional*: "A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement." [9]
3. *RNF arquitectónico*: Especifica una necesidad que impacta en la *solución arquitectónica* (δ-8) del sistema software; p. ej., un NFR que determine la distribución de los componentes tal como: "el sistema debe estar disponible 24 horas por día, 7 días por semana", este tipo de requisito implicaría replicación de varios *componentes arquitectónicos* (δ-11).

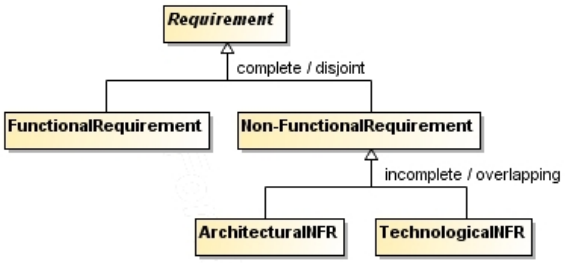

Fig. 2: Diagrama de clases correspondiente a los requisitos

4. *RNF tecnológico*: Especifica una necesidad que impacta en la selección de *tecnologías* (δ-13) del sistema software; p. ej., un NFR que imponga el uso de una tecnología concreta ("el nuevo sistema debe integrarse con la base de datos del sistema actual"). Puede suceder que un RNF sea a la vez arquitectónico y tecnológico.

Destacamos que en ArchiTech nos focalizamos en los NFRs que nos permiten tomar decisiones arquitectónicas o tecnológicas, que se obtienen mediante un proceso de refinamiento de los objetivos iniciales de alto nivel (p. ej., "alta eficiencia"), usando típicamente metodologías orientadas a objetivos [10]. Estos procesos de refinamiento quedan fuera de los objetivos de este artículo. Los NFRs no tienen porqué llegar a ser operacionales, es más, en algunos casos un exceso de detalle puede dificultar el proceso de decisión.

Estilos. Los estilos definen un esquema global y reusable que explica cómo deben construirse los artefactos. Diferenciamos entre estilo arquitectónico y estilo tecnológico (v. Fig. 3).

5. *Estilo arquitectónico*: Un estilo arquitectónico, según M. Shaw y D. Garlan [11] y Buschmann *et al.* [12] incluye la descripción de los *tipos de componentes arquitectónicos* (δ-10) y su organización, así como la interacción entre los mismos Ejemplos habituales son: estilo SOA, estilo en capas, etc. Un estilo arquitectónico puede especializarse en otros; p. ej., el estilo en 3 capas es una especialización del estilo en capas.
6. Para mejorar la reusabilidad de los estilos arquitectónicos, en esta ontología se propone el concepto de *variación del estilo*. A diferencia de una especialización, una variación del estilo afecta a una parte parcial del estilo, pudiéndose combinar más de una variación para formar nuevos estilos (v. Sección 4 para ejemplo).
7. *Estilo tecnológico*: No existe una definición en la literatura académica, pero siguiendo el patrón de los *estilos arquitectónicos* (δ-5), los estilos tecnológicos se representan por una descripción de los *roles tecnológicos* (δ-12) que los forman. Ejemplos habituales son: "Stack solution", Java y .NET. Los estilos tecnológicos pueden establecer *limitaciones* (en la Fig. 3 y 4 están representadas por la clase "Restriction") sobre las *tecnologías* (δ-13) que implementan el conjunto de roles tecnológicos. Las tecnologías válidas para un estilo tecnológico han de ser capaces de funcionar conjuntamente. Un estilo tecnológico puede especializarse; p. ej., el estilo LAMP es una especialización del estilo "Stack solution". Igual que antes, se contemplan *variaciones de estilos* (δ-6) para los estilos tecnológicos.

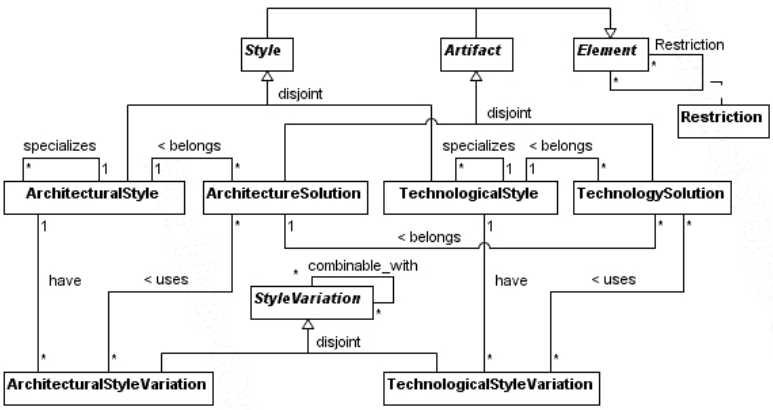

Fig. 3: Diagrama de clases correspondientes a los estilos y los artefactos

Artefactos. Un artefacto es una pieza de información concreta del sistema software en desarrollo. Consideramos como artefactos la solución arquitectónica y la solución tecnológica (v. Fig. 3). Los artefactos deben seguir las pautas marcadas por el estilo al que pertenecen.

8. *Solución arquitectónica*: A este término muchas veces se le llama simplemente *arquitectura* o *arquitectura software*, pero ello puede provocar confusiones con el término *estilo arquitectónico* (δ-5). "A software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system. The software architecture of a system is an artifact. It is the result of the software design activity." [12]. Una solución arquitectónica está construida siguiendo las pautas impuestas por el estilo arquitectónico al cual pertenece.
9. *Solución tecnológica*: Siguiendo la definición de *solución arquitectónica* (δ-8), definimos solución tecnológica como la descripción de las *tecnologías* (δ-13) y sus relaciones, así como el papel que juegan dentro del sistema software. Esta solución debe contemplar las tecnologías necesarias para cubrir todos los *roles tecnológicos* (δ-12) exigidos por el *estilo tecnológico* (δ-7) al que pertenece. P. ej. Oracle 11g para el rol de SGBD, Ubuntu 9.0 para el de SO, PHP 4.3 para el de lenguaje de programación, etc.

Modelos. En este caso consideramos los tres tipos habituales de modelos en el DSDM: *CIM*, *PIM* y *PSM* y nos adherimos a las definiciones habituales de los mismos. No obstante, el papel que tendrían estos modelos en architech seria: el CIM representa el dominio (diagramas de casos de uso, diagramas de clases, etc.), el PIM representa la arquitectura (diagramas estructurados en componentes mostrando sus relaciones), y el PSM representa la tecnología (diagramas adaptados para generar código para un conjunto de tecnologías específicas).

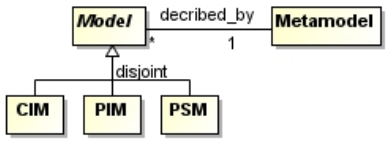

Fig. 4: Diagrama de clases correspondientes a los modelos

Para que la ontología sea autocontenida incluimos las siguientes definiciones de la guía MDA [13] (v. Fig. 4).

- Model: "A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language".
- Metamodel: "A description or definition of a well-defined language in the form of a model." [14].
- Computation Independent Model (CIM): "A CIM is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure

of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification".

- Platform Independent Model (PIM): "A PIM is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type".
- Platform Specific Model (PSM): "A PSM is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform".

Para poder enlazar los cuatro grupos de conceptos principales han surgido varios conceptos mediadores que detallamos a continuación:

Componentes de los estilos y de los artefactos. Tanto los estilos como los artefactos son elementos que se componen de otros elementos, los componentes. Los componentes pueden componerse a su vez de otros componentes de su mismo tipo. Diferenciamos entre cuatro tipos de componentes (v. Fig. 5).

10. *Tipo de componente arquitectónico*: Un tipo de componente arquitectónico es una pieza de un *estilo arquitectónico* (δ-5), a su vez define como son los *componentes arquitectónicos* (δ-11). Los tipos de componente arquitectónicos se pueden especializar; p. ej., la capa es un tipo de componente arquitectónico que se puede especializar en un segundo tipo, la capa de persistencia.
11. *Componente arquitectónico*: Un componente arquitectónico es una realización de un *tipo de componente arquitectónico* (δ-10) que forma parte de la solución arquitectónica (δ-8) de un sistema software determinado. P. ej., el componente de gestión de usuarios de Easychair podría ser del tipo de componente arquitectónico "módulo" (que suponemos forma parte del estilo arquitectónico de Easychair). Cada componente arquitectónico del sistema software juega un *rol concreto* para cada una de los *roles tecnológicos* (δ-12) requeridos por el *tipo de componente arquitectónico* al cual pertenece. Estos *roles concretos* nos permiten seleccionar la *tecnología* (δ-13) más adecuada a nivel de componente arquitectónico.

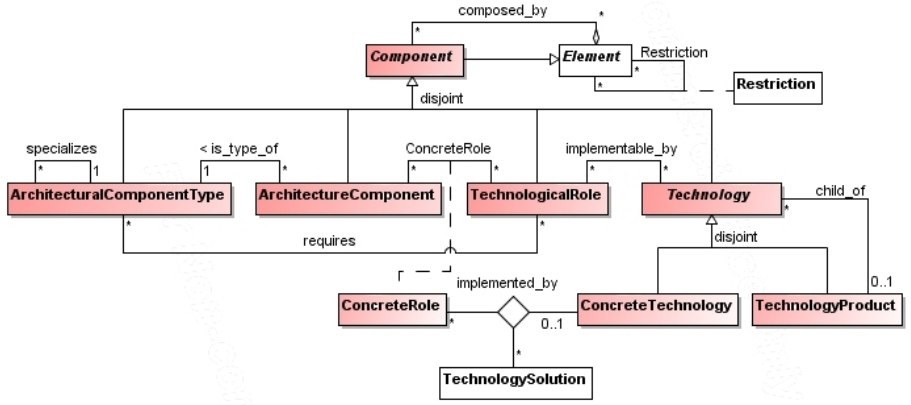

Fig. 5: Diagrama de clases de los componentes de los estilos y los artefactos

12. *Rol tecnológico*: Un rol tecnológico es un tipo de *tecnología* (δ-13). P. ej., el SGBD, o el servidor Web juegan un rol dentro de un *estilo tecnológico* (δ-7). Estos roles tecnológicos aparecen por necesidades de los *tipos de componentes arquitectónicos* (δ-10), por formar parte del *estilo tecnológico*, o por ser dependencia de otro rol tecnológico. Aunque no todos los roles tecnológicos, ni las tecnologías que los implementan, jueguen un papel en la construcción del PSM, es necesario tenerlos todos para poder asegurar que pueden funcionar conjuntamente.
13. *Tecnología*: Una tecnología es una pieza de software que ofrece unas facilidades para la implementación parcial o completa de alguna funcionalidad del sistema software. Una tecnología puede referirse a un *producto tecnológico* o bien a una *tecnología concreta* de un producto tecnológico; p. ej., Oracle es un producto y Oracle 11g es una tecnología concreta. Las tecnologías concretas se usan para construir *soluciones tecnológicas* (δ-9) mientras que la finalidad de los productos tecnológicos es poder establecer *limitaciones* para todas las versiones de un producto; p. ej., la especificación de incompatibilidades entre productos tecnológicos.

Restricciones derivadas de los requisitos. Dado que los NFR del sistema suelen expresarse de diversas formas (p. ej,. lenguaje natural, Volere, etc.) es necesario traducirlos a una nomenclatura que nos permita trabajar con ellas. La nomenclatura escogida para ArchiTech son las restricciones (en el diagrama de la Fig. 6 aparecen como "Constraints").

14. Una *restricción* es una expresión evaluable formada por un conjunto de condiciones sobre las *propiedades* (δ-15). Idealmente esta expresión puede contener operaciones booleanas entre las diversas condiciones y condiciones comparativas sobre las propiedades (v. ejemplo en la sección 4). Se contemplan dos tipos de restricciones, arquitectónicas y tecnológicas. Respectivamente, cada tipo de restricción condiciona a un tipo de propiedades.

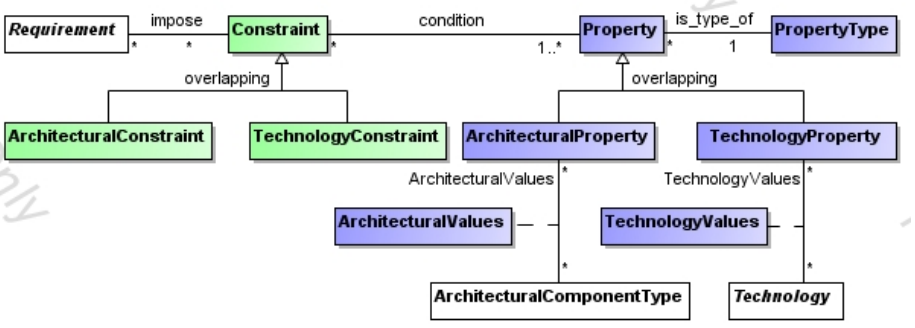

Fig. 6: Diagrama de clases correspondientes a las restricciones y propiedades

Propiedades. Algunos componentes (los *tipos de componentes arquitectónicos* (δ-10) y las *tecnologías* (δ-13)) tienen propiedades que permiten determinar cuando es mejor usar uno u otro por medio de la evaluación de las *restricciones* (δ-14). V. Fig 6.

15. Las *propiedades* son características comunes a diversos elementos alternativos. Los componentes con propiedades establecen valores para estos. (p. ej., la propiedad de poder trabajar de forma distribuida).
16. Los *tipos de propiedades* son descriptores del dominio de valores validos para una *propiedad* (δ-15), p. ej., números enteros, cadenas de caracteres, conjunto de valores definidos, etc. También se considera como un tipo de propiedad u conjunto de valores (p. ej, propiedades multivaluadas).

Transformaciones. Las transformaciones de modelos son mecanismos que nos permiten hacer evolucionar un modelo a otro. Las transformaciones se especifican usando un lenguaje de transformación. Diferenciamos entre transformaciones CIM a PIM o transformaciones arquitectónicas, y transformación PIM a PSM o transformaciones tecnológicas. Los dos artefactos descritos anteriormente contemplan, respectivamente, el conjunto de transformaciones necesarias para obtener el PIM y el PSM (v. Fig. 7).

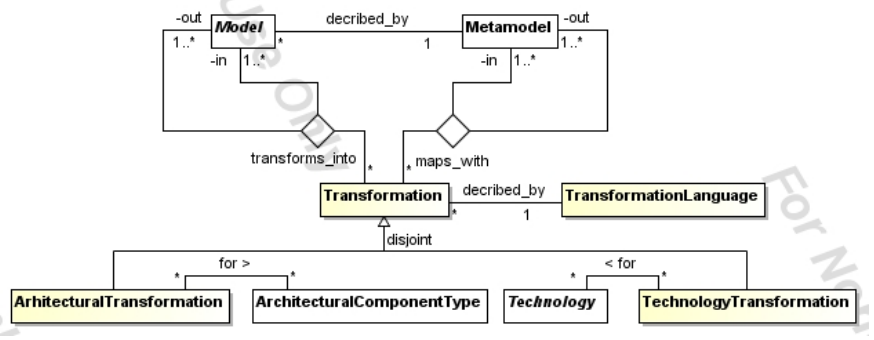

Fig. 7: Diagrama de clases correspondientes a las transformaciones

4 Escenario de Uso

El escenario de uso propuesto está ideado para que podamos constatar cómo los RNF pueden afectar de una forma muy significativa al proceso de desarrollo de un sistema software, incluso partiendo de un mismo conjunto de requisitos funcionales. El escenario escogido para ejemplificarlo es un sistema software para la gestión de conferencias (SGC). Se plantean dos posibles alternativas para este mismo escenario:

- Alternativa A: Una prestigiosa editorial quiere desarrollar el SGC e integrarlo en su sistema de información para asegurar la calidad de los artículos que publica. El SGC será usado por personas sin conocimientos informáticos. Se ha impuesto un tope de 100.000€ para financiar el proyecto y un plazo de 3 meses para terminarlo.
- Alternativa B: Una gran cantidad de conferencias están siendo gestionadas por universidades españolas. Por esta razón se ha propuesto un proyecto universitario coordinado para desarrollar un SGC de ámbito español. Cada universidad desea guardar los datos de las conferencias que organiza. Se espera que los estudiantes que implementan aprendan las tecnologías necesarias durante el transcurso del proyecto. Los costes serán sufragados por las universidades participantes.

Para cada alternativa, la editorial (A) y los estudiantes (B) han extraído un conjunto de requisitos distintos aplicando metodologías orientadas a objetivos [10]. Sólo nos centramos en los NFR que pueden tener consecuencias arquitectónicas o tecnológicas. Podemos ver el resultado de ambas alternativas en la Tabla 1.

Podríamos llegar a las siguientes conclusiones de forma intuitiva:

- Ambas alternativas parecen adecuadas para un estilo arquitectónico en tres capas debido a que se trata de un sistema de información transaccional. En consecuencia la solución arquitectónica resultante sería similar en los dos casos.
- La alternativa A sugiere que sea una aplicación de escritorio. Este tipo de aplicaciones son más habituales y fáciles de usar. Por otro lado, en la alternativa B, parece más adecuado pensar en una aplicación Web, ya que para éstas existen muchas más tecnologías de software libre y la mayoría tienen mucho soporte.
- El estilo tecnológico en el primer caso viene determinado por el escenario: .Net, en el segundo caso, probablemente optaríamos por LAMP reforzando así la idea de usar software libre en el entorno Web.
- Dependiendo de la experiencia como desarrollador algunas decisiones serían más claras que otras. Por ejemplo, respecto a la gestión de los datos hay infinidad de opciones, desde ficheros de texto plano hasta el uso de "data mappers". En la alternativa A no hay mucho que pensar, el escenario nos impone el uso de un SGBD concreto y posiblemente la mayoría de desarrolladores aprovecharían las facilidades para la gestión de datos que ofrece ADO.Net. En contrapartida, la segunda alternativa es menos clara. La necesidad de gestionar diversas bases de datos desde un mismo sistema nos obliga ha conocer mucho mejor los detalles y posibilidades de cada opción tecnológica, no seria una elección fácil ni intuitiva (exceptuando si se tiene experiencia previa en un sistema similar).

Como se puede observar a partir de estas conclusiones, los RNF son cruciales para determinar de forma precisa tanto el estilo arquitectónico como el tecnológico. Además los RNF pueden dirigirnos hacia sistemas software completamente distintos.

Tabla 1. Requisitos extraídos en cada alternativa

	Alternativa A	**Alternativa B**
R1	El sistema debe ser desarrollado con .Net versión 2.0	El sistema debe ser desarrollado con software libre
R2	La interfaz del usuario debe ser simple e intuitiva	Las tecnologías usadas deben tener un buen soporte de la comunidad
R3	Los datos deben guardarse en el SGBD de la editorial (Oracle 11g)	Los datos de cada conferencia deben guardarse en la universidad organizadora

Tomando como partida el escenario y la alternativa B, a continuación exploraremos los detalles de cómo se llegaría a una conclusión similar, usando en esta ocasión el razonamiento de ArchiTech. En el momento de tomar decisiones, ArchiTech tiene dos principios a seguir: el primero es que siempre que hayan especializaciones priorizará la opción más especifica, y el segundo es que si más de una opción cumple todos los NFR, será el usuario quien tendrá la ultima palabra. La necesidad de los conceptos descritos en la ontología se verá reflejada en este ejemplo.

Partiendo de los requisitos de la Tabla 1 el motor de Architech detectaría las siguientes restricciones en función de varias propiedades:

- C1 - La propiedad “Persistencia” debe incluir el valor “distribuida”.
- C2 - La propiedad “Tipo de licencia” debe ser igual a “OSS”.

C1 es una restricción arquitectónica derivada de R3-B (Tabla 1) y C2 es una restricción tecnológica derivada de R1-B. Por supuesto que en un escenario real aparecerían muchos más requisitos y en consecuencia muchas más restricciones. Para que el ejemplo sea fácil de entender nos centraremos sólo en estas dos restricciones.

Una vez tenemos las restricciones del sistema software empezaremos por educir el estilo arquitectónico. Supongamos que el marco de trabajo conoce únicamente los siguientes estilos arquitectónicos:

- AS1 - “En capas”
- AS2 - “3-Capas”
- AS3 - “Pipes and filters”
- AS4 - “Blackboard”

Educimos primeramente AS2, usando las pautas descritas en [12]. AS2 es una especialización de AS1. Los tipos de componentes arquitectónicos de AS1 son: módulo y capa. Una capa esta compuesta por módulos. Los tipos adicionales necesarios para AS2 son: capa de presentación, capa de dominio y capa de persistencia. Todos son especialización del tipo “capa”. Además AS2 incluye las limitaciones habituales en una arquitectura en 3-capas, expresadas mediante restricciones: la comunicación entre capas así como el número de capas y su tipo.

El marco de trabajo contempla dos posibles variaciones para AS2: persistencia distribuida y persistencia local. Como es lógico, debido a la restricción C1 se escogerá la variación de persistencia distribuida. Ésta usa dos especializaciones de los componentes: capa de persistencia distribuida y módulo distribuido.

Una vez sabemos cual será el estilo arquitectónico podemos iniciar la construcción de la solución arquitectónica. Para ello se usara el modelo CIM y las reglas de

construcción del estilo para separar la funcionalidad en diversos módulos y capas. Cuando se haya determinado cómo se compone la arquitectura del sistema software se aplicarán las transformaciones necesarias para obtener el PIM.

Para la parte tecnológica del proceso, supongamos que ArchiTech conoce los siguientes estilos tecnológicos:

- TS1 - “Stack solution”
- TS2 - “LAMP”
- TS3 - “Java”
- TS4 - “.Net”

TS2 es una especialización de TS1. TS1 requiere cuatro roles tecnológicos: “SGBD”, “Lenguaje de programación”, “Servidor Web” y “Sistema operativo”. Los tipos de componentes arquitectónicos del estilo educido (AS2+persistencia distribuida) requieren cinco roles tecnológicos: “Tecnología de presentación”, “Lenguaje de programación”, “Gestor de transacciones”, “Tecnología de autenticación” y “Data mapper”. TS2 no añade roles tecnológicos, pero sí limitaciones: El “Sistema operativo” debe ser “Linux”, el “Servidor Web” debe ser “Apache”, etc. Los componentes de persistencia distribuidos imponen la limitación de que la tecnología que los implemente deberá tener la capacidad de funcionar de forma distribuida.

En este ejemplo TS2 seria educido como el estilo tecnológico porque las tecnologías que requiere son OSS. Como se ha dicho antes, este ejemplo ha sido simplificado. En un caso real habrían muchas más restricciones que satisfacer.

La solución tecnológica sería el conjunto de tecnologías concretas seleccionadas (aplicando las limitaciones de compatibilidad y otras restricciones) para cada uno de los componentes arquitectónicos. P. ej., el módulo de la capa de presentación encargado del envío de artículos se podría asociar con la tecnología “PHP 5.0”. Un posible razonamiento para esta selección podría ser que a partir de la versión 5, PHP soporta el uso de excepciones las cuales son adecuadas para un sistema transaccional.

5 Conclusiones y Trabajo Futuro

En este artículo se han identificado, descrito y relacionado todos los conceptos que han ido apareciendo en el marco de trabajo que se esta desarrollando, ArchiTech, para el tratamiento de NFR en el proceso de DSDM.

Existen otros trabajos que fomentan la introducción de los RNF en los procesos de DSDM. De todos ellos queremos destacar la aportación de V. Cortellessa [5], en su trabajo hacen un uso de los RNF paralelo al proceso DSDM, existiendo de esta forma modelos CIM, PIM y PSM para cada gran grupo de RNF (p. ej., requisitos de eficiencia), en el articulo estos requisitos son llamados CINFR, PINFR y PSNFR, haciendo referencia al nivel de abstracción de éstos pero no diferencian entre NFR arquitectónicos y NFR tecnológicos. Esto obliga a contemplar un gran número de modelos y de transformaciones adicionales. Por el contrario, en nuestra aportación los RNF juegan un papel decisional y ayudan a dirigir el proceso DSDM.

Consideramos que el trabajo realizado es un buen punto de partida para desarrollar, en un futuro próximo, un prototipo de ArchiTech. Principalmente el propósito del prototipo será validar los conceptos descritos en este artículo en relación a la integración de los RNF dentro del DSDM. Con referencia a los aspectos ontológicos de la propuesta, una vez la propuesta de este artículo se consolide, será preciso complementarla con extensiones que cubran los conceptos principales de ArchiTech: ontología para los RNF, para los estilos arquitectónicos, para las tecnologías, etc. Algunas serán más dinámicas (p. ej., la citada para tecnologías) mientras que otras evolucionarán más lentamente (p. ej., las referentes a estilos arquitectónicos).

References

1. Glinz, M.: On Non-Functional Requirements. In: 15th IEEE Int. Requirements Engineering Conf., Delhi, India (2007)
2. Brooks, F. P.: No Silver Bullet - Essence and Accidents of Software Engineering. In: IEEE Computer 20, 4, pp. 10-19 (April 1987)
3. Finkelstein, A., and Dowell J.: A comedy of errors: the London Ambulance Service case study. In: 8th Int. Workshop on Software Specification and Design (IWSSD), Germany (1996)
4. Norma internacional: ISO 9000:2000: Sistemas de gestión de la calidad - Conceptos y vocabulario, traducción certificada (2000)
5. Cortellessa, V., Marco, A. D., and P. Inverardi: Non-functional Modeling and Validation in Model-Driven Architecture. In: Working IEEE/IFIP Conf. on Software Architecture (WICSA). Mumbai, India (2007)
6. Ardagna, D., Ghezzi, C., and Mirandola, R.: Rethinking the Use of Models in Software Architecture. In: Int. Conf. Series on the Quality of Software Architectures (QoSA). Karlsruhe, Germany (2008)
7. Gruber, T. R.: Towards principles for the design of ontologies used for knowledge sharing. In: N. Guarino & R. Poli, eds, 'Formal Ontology in Conceptual Analysis and Knowledge Representation', Amsterdam, Nederland (1994)
8. Dorfman, M., and Thayer, R. H.: Standards, Guidelines and Examples: System and Software Requirements Engineering. In: IEEE Computer Society Press, Los Alamitos, CA (1990)
9. Jacobson, I., Booch, G., and Rumbaugh, J.: The Unified Software Development Process. Reading, Mass.: Addison Wesley (1999)
10. Lamsweerde, A. van: Goal-Oriented Requirements Engineering: A Guided Tour. In: 5th IEEE Int. Symp. on Requirements Engineering (RE), Toronto, Canada (2001).
11. Shaw, M., and Garlan, D., Software architecture: perspectives on an emerging discipline, Prentice Hall, Upper Saddle River (1996)
12. Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., and Stal, M.: Pattern-Oriented Software Architecture. A system of Patterns, John Wiley & Sons (1996)
13. Vallecillo, A.: Modelado de aplicaciones software mediante MDA. In the course: Servicios Avanzados Basados en Componentes. http://www.lcc.uma.es/~canal/sabc/MDA-doctorado.pdf (2008)
14. Kleppe, A., Warmer, J., and Bast, W.: MDA Explained, The Model Driven Architecture: Practice and Promise: Addison-Wesley, 2003.

ANEXO: Diagrama conceptual en una pieza